8-3-2020

End-to-End Structure-Aware Convolutional Networks on Graphs

Chao Shang
University of Connecticut - Storrs, chao.shang@uconn.edu

End-to-End Structure-Aware Convolutional Networks on Graphs

Chao Shang, Ph.D.

University of Connecticut, 2020

ABSTRACT

Convolutional neural networks (CNNs) are powerful tools to model data of a grid-like structure, such as image, video, and speech. However, a broad range of scientific problems generate data that naturally lie in irregular grids with non-Euclidean metrics, such as knowledge graphs, molecular graphs, and traffic networks. The generalization of CNNs to non-Euclidean structured data such as graphs is not straightforward. The classical convolutions cannot be applied directly to graphs, due to the lack of global parameterization, a common system of coordinates, and shift-invariance properties.

In this dissertation, we propose several structure-aware convolutional neural network models to calculate graph convolutions efficiently over both small-scale and large-scale graphs. The proposed networks can be trained by an end-to-end training method where a stochastic gradient descent algorithm back-propagates over all network components rather than a stage-wise training scheme where the different components are tuned separately. These models are built for exploring the structure information to improve many prediction tasks: (1) The first part of the dissertation focuses on a large-scale knowledge graph. Our new approach learns the graph connectivity structure so it can infer new edges in a knowledge graph and grow an

input knowledge graph to be more complete. This model not only utilizes the node (or entity) attributes and edge relations in a knowledge graph but also preserves the so-called translational property between entities and relations. (2) The second part extends the convolution operation to small-scale hydrogen-depleted molecular graphs. Unlike the first model that learns from a single graph of massive size, this method learns a novel graph-based model from a massive amount of small graphs. We propose a consistent edge-aware graph convolutional network that determines consistent edge attentions to the same type of edges appearing in different molecular graphs and predicts a molecule's properties. This model exploits the general consistency of the bond energies and bond lengths across various molecular graphs. (3) The third part focuses on the exploration of the correlation and causation among multivariate time series in a graph where a node represents a time series variable and an edge indicates if a time series causes another time series. When the connectivity (edge) structure is not available or incomplete, we propose a discrete structure learning method that learns the hidden graph structure simultaneously when constructing the predictive model for the time series data. Extensive experiments demonstrate the advantages of the proposed techniques over the state of the art in knowledge graph completion, molecular quantitative structure-activity relationship prediction, and multivariate time series forecasting.

End-to-End Structure-Aware Convolutional Networks on Graphs

Chao Shang

M.S., Beijing University of Posts and Telecommunications, 2015

B.S., Shandong Jianzhu University, 2012

A Dissertation

Submitted in Partial Fulfillment of the

Requirements for the Degree of

Doctor of Philosophy

at the

University of Connecticut

2020

APPROVAL PAGE

Doctor of Philosophy Dissertation

End-to-End Structure-Aware Convolutional Networks on Graphs

Presented by

Chao Shang, B.S., M.S.

Major Advisor _____
Jinbo Bi

Associate Advisor _____
Alexander Russell

Associate Advisor _____
Yufeng Wu

University of Connecticut

2020

ACKNOWLEDGMENTS

First and foremost I would like to express my deep gratitude to my advisor, professor Jinbo Bi for her dedicated support and guidance. I am extremely grateful to be her Ph.D. student. I appreciate all her contributions of time, ideas, and encouragement to make my Ph.D. experience productive and stimulating. Professor Bi has given me the freedom to pursue various research projects and provided insightful discussions and scientific advice about the research. No matter what difficulty I meet, she always supports and encourages me. I am also thankful for the excellent example she has provided as a successful scientist and professor.

I would like to thank the members of my doctoral committee, Dr. Alexander Russell and Dr. Yufeng Wu, for their encouragement and insightful comments. I appreciate their great suggestions on my research that enlightened me the first glance of research. I give my gratitude to Dr. Derek Aguiar and Dr. Caiwen Ding who serve my review committee members. Dr. Aguiar provided many valuable suggestions on my thesis and Dr. Ding taught me new domain knowledge in some potential research directions. And I thank Dr. Fei Wang to be my advisor for about a year. I am grateful for his support and inspirational discussions. I would also like to thank Dr. Jing Huang and Dr. Jie Chen for offering me the internship opportunities and working together on diverse and exciting projects.

My sincere thanks also go to Jiangwen Sun, Jin Lu, Tingyang Xu, Chunjiang Zhu, Guannan Liang, Qianqian Tong, Aaron Palmer, Tan Zhu, Qinqing Liu, Fei Dou,

Xinyu Wang, Zhenxiang Gao, Guoqing Chao, Ko-Shin Chen and Xin Wang for the insightful discussions and for an enjoyable time in the past five years. Especially, I would like to thank Jiangwen and Jin for invaluable guidance and assistance in my research.

Lastly, my deep and sincere gratitude to my family for their continuous love, help, and support. I would like to express my deepest appreciation for my Dad and Mom for their understanding, support, encouragement, and love. They are my spiritual support throughout my life. I love them so much, and I would not have made it this far without them. And I would like to give my deepest love and gratitude to my wife, Xue Wu. She is my faithful support during this amazing journey. She cherishes with me every great moment and supports me whenever I needed it. Wish my family health and happiness forever.

List of Figures

List of Tables

Contents

Chapter 1

Introduction

Convolutional neural networks (CNNs) [47] have been widely used in many applications to model grid-like structure data, e.g., image, video, and speech. On a regular grid, CNNs offer an efficient way to extract local stationary structures and features that are shared across the objects (e.g., images) [46]. However, a broad range of scientific problems generate data that naturally lie in irregular spaces with non-Euclidean metrics, such as molecular graphs in computational chemistry, knowledge graphs, traffic networks, and telecommunication networks. Data in these areas are usually represented as graphs that encode complex geometric structures with node and edge attributes. The generalization of CNNs to graph inputs is not straightforward. For example, in an image represented on a 2-dimensional Euclidean grid, a kernel from CNNs can move left, right, etc. However, graphs are non-euclidean, where the movements of a kernel don't have any meaning. Due to the lack of global parameterization, a common system of coordinates, vector space structure, or shift-invariance properties [8], the classical convolutions which use fixed filter size and stride distance cannot be

applied directly to graphs that have arbitrary structures.

The task of this dissertation is to design a new type of neural networks to work directly on graph-structured data from irregular domains. By designing novel neural networks on arbitrary graph structures, we directly leverage their structural information to enhance prediction tasks. Recently graph convolutional networks (GCNs), also known as graph neural networks (GNNs), have been developed and successfully solved tasks [60] such as manifold analysis [63], predictions of user preference and connectivity in social network [9], node classification [30, 44, 28], and generation of fingerprints from molecular graphs [25].

In a similar spirit to CNNs, GCN methods aggregate neighboring information based on the connectivity of the graph through filters. The aggregation is often carried out either in a spatial way [18, 67, 39, 92, 30] or a spectral way [10, 15, 44, 13, 22]. In the spatial way, the graph convolution operation is similar to regular convolution from CNNs. The spatial GCNs make convolutions by aggregating node information from the neighbors into the corresponding central node. In spectral GCNs, node attributes are viewed as graph signals, and the convolution operation is defined through Fourier transform which is derived from graph Laplacian and its eigenspace. The localized convolutional filters extracted local features independently motivate the spatial graph convolutions via first-order approximation of spectral graph convolutions. Section 3.2 provides more comparisons of these GCN approaches. In addition, there are many surveys [90, 100, 26] that have given more details of the GCNs. However, existing GCN methods have many limitations or haven't been used in handling the graph structured data from many applications. For example, current models cannot take full advantage of the characteristics of chemical bonds in molecular graphs. Hence we present multiple models to explore the structure information in multiple domains.

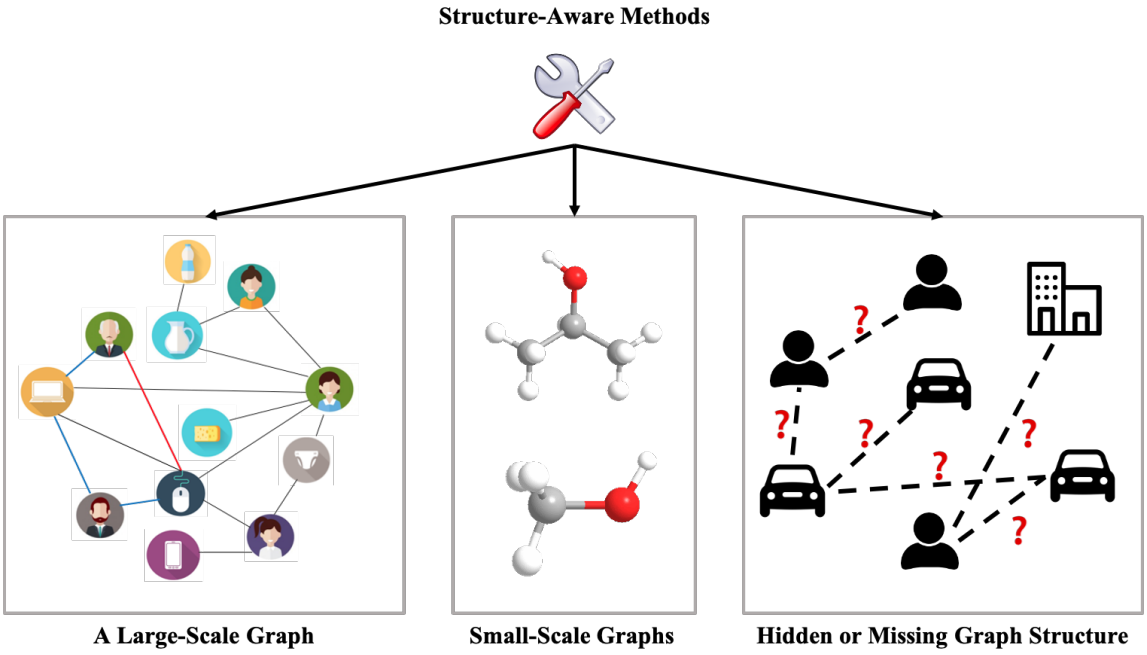


Figure 1.1: Structure mining by designing structure-aware methods.

In this dissertation, we propose structure-aware convolutional neural networks that will explore the structure information from three aspects [77, 76] as shown in Figure 1.1: (1) For a large-scale knowledge graph, we take advantage of graph structure and attributes of graph nodes to solve knowledge graph completion task; (2) For the small-scale molecular graphs, we explore the common structural characteristics of multiple graphs to improve molecular quantitative structure-activity relationship prediction; (3) Since graph structure is not always available or it may be incomplete, we propose a discrete structure learning model to explicitly learn the pairwise interactions in the form of a graph and use it to improve forecasting accuracy.

Firstly, large-scale knowledge bases (KBs), such as Freebase [6], DBpedia [4] and YAGO3 [58], have been built to store structured information to support many applications. KBs are multi-relational graphs whose nodes represent entities and edges

represent relationships between entities, and the edges are labeled with different relations. These KBs are extensively used for web search, recommendation, and question answering. Although these KBs have already contained millions of entities and triplets, they are far from complete compared to existing facts and newly added knowledge of the real world. Therefore knowledge base completion is important to predict new triplets based on existing ones and thus to further expand KBs. For solving this task, we propose a novel end-to-end Structure-Aware Convolutional Network (*SACN*). *SACN* not only utilizes knowledge graph node structure, node attributes and edge relation types, but also keeps the translational characteristic [7] between entities and relations. We demonstrate the effectiveness of the proposed *SACN* on standard FB15k-237 and WN18RR datasets, and it gives about 10% relative improvement over the state-of-the-art *ConvE* in terms of HITS@1, HITS@3, and HITS@10.

Secondly, although GCNs that extend the convolution operation from images to graphs have led to competitive performance, the existing GCNs are still difficult to handle the chemical problems. Recently multiple GCNs are applied to the compound structures which are represented by the hydrogen-depleted molecular graphs with different sizes. The GCNs built for binary connectivity matrices do not account for the edge consistency in multiple molecular graphs, i.e., the consistency of bond energies (enthalpies) and bond lengths (interatomic distances) even across the edges in different molecular graphs. In this dissertation, we propose a variant of GCN where a molecular graph is first decomposed into multiple views of the graph, each comprising a specific type of edges. In each view, an edge consistency constraint is enforced that similar edges in different graphs can receive similar attention weights when passing information. We also prove that in each layer, our method corresponds to a spectral filter derived by the first order Chebyshev approximation of graph Laplacian. Exten-

sive experiments demonstrate the substantial advantages of the proposed technique in quantitative structure-activity relationship prediction.

Thirdly, if an explicit graph structure is known or given, GCN methods are powerful tools to exploit the structure to improve the predictions. However, the graph structure is missing or incomplete in some situations, such as multivariate time series (MTS) data. Here we attempt to learn the hidden or missing graph. Exploration of the correlation and causation among the variables in a multivariate time series shows promise in enhancing the predictive performance of a time series model. When using deep neural networks as forecasting models, we hypothesize that exploiting the pairwise information among multiple (multivariate) time series improves their simultaneous forecasting. In this dissertation, we propose a discrete structure learning approach that learns a hidden graph structure when constructing the predictive model, if an explicit one is unknown. We cast the problem as learning a probabilistic graph model by optimizing the mean performance over a graph distribution. We first learn the graph distribution which is parameterized by a neural network so that discrete graphs can be sampled differentiably through reparameterization. Empirical evaluations compare the proposed method with a recently proposed bilevel learning approach, showing that our method is simpler, more efficient, and better performing.

Chapter 2

Structure-Aware Learning on a Large-Scale Knowledge Graph

2.1 Motivation

For understanding the real-world entities and their relationships, the knowledge graph is first proposed by Google to significantly enhance the value of information returned by Google searches. In a knowledge graph, nodes represent entities, edges represent relationships between entities, and the edges are labeled with different relations. The relationships are organized in the forms of (s, r, o) triplets (e.g., entity s = Abraham Lincoln, relation r = DateOfBirth, entity o = 02-12-1809). Since those knowledge graphs are far from complete compared to existing facts and newly added knowledge, knowledge base completion is an important task that predicts new relational facts between entities under the supervision of the existing knowledge graph.

One of the recent active research areas for knowledge base completion is knowledge

graph embedding: it encodes the semantics of entities and relations in a continuous low-dimensional vector space (called embeddings). These embeddings are then used for predicting new relations. Starting from a simple and effective approach called *TransE* [7], many knowledge graph embedding methods have been proposed, such as *TransH* [87], *TransR* [54], *DistMult* [93], *TransD* [37], *ComplEx* [83], *STransE* [66]. Some surveys [65, 86] give details and comparisons of these embedding methods.

The most recent *ConvE* [16] model uses 2D convolution over embeddings and multiple layers of nonlinear features, and achieves state-of-the-art performance on common benchmark datasets for knowledge graph link prediction. In *ConvE*, the embeddings of two entities s and r are reshaped and concatenated into an input matrix and fed to the convolution layer. Convolutional filters of size $n \times n$ are used to output feature maps that are across different dimensional embedding entries. The model can be efficiently trained and scales to large knowledge graphs. However, *ConvE* does not keep the translational property as *TransE* which is an additive embedding vector operation [64]: $e_s + e_r \approx e_o$. In addition, there is no structure enforcement in the embedding space of *ConvE*. In this dissertation, we remove the reshape step of *ConvE* and operate convolutional filters directly in the same dimensions of s and r. This modification gives better performance compared with the original *ConvE*, and has an intuitive interpretation which keeps the global learning metric the same for s, r, and o in an embedding triple (e_s, e_r, e_o). We name this embedding as *Conv-TransE*.

ConvE also does not incorporate connectivity structure in the knowledge graph into the embedding space. In contrast, graph convolutional networks have been effective tools to create node embeddings that aggregate local information in the graph neighborhood for each node [44, 30, 42, 70, 76]. GCN models have additional benefits [31], such as leveraging the attributes associated with nodes. They can also impose

the same aggregation scheme when computing the convolution for each node, which can be considered a method of regularization, and improves efficiency.

In this dissertation, we propose an end-to-end graph Structure-Aware Convolutional Network (*SACN*) model that integrates the benefits of GCN and *ConvE*. *SACN* consists of an encoder of a weighted graph convolutional network (*WGCN*), and a decoder of a convolutional network called *Conv-TransE*. *WGCN* utilizes knowledge graph node structure, node attributes and relation types. It has learnable weights to determine the amount of information from neighbors used in local aggregation, leading to more accurate embeddings of graph nodes. Node attributes are added to *WGCN* as additional for easy integration. The output of *WGCN* becomes the input of the decoder *Conv-TransE*. *Conv-TransE* is similar to *ConvE* but with the difference that *Conv-TransE* keeps the translational characteristic between entities and relations. We show that *Conv-TransE* performs better than *ConvE*, and our *SACN* improves further on top of *Conv-TransE* in the standard benchmark datasets.

Our contributions are summarized as follows:

- We present an end-to-end network learning framework *SACN* that takes benefit of both GCN and *Conv-TransE*. The encoder GCN model leverages graph structure and attributes of graph nodes. The decoder *Conv-TransE* simplifies *ConvE* with special convolutions and keeps the translational property of *TransE* and the prediction performance of *ConvE*;

- We demonstrate the effectiveness of our proposed *SACN* on the standard FB15k-237 and WN18RR datasets, and show about 10% relative improvement over the state-of-the-art *ConvE* in terms of HITS@1, HITS@3, and HITS@10.

2.2 Related Work

Knowledge graph embedding learning has been an active research area with applications directly in knowledge base completion (i.e. link prediction) and relation extractions. TransE [7] started this line of work by projecting both entities and relations into the same embedding vector space, with translational constraint of $e_s + e_r \approx e_o$. Later works enhanced KG embedding models such as TransH [87], TransR [54], and TransD [37] introduced new representations of relational translation and thus increased model complexity. These models were categorized as *translational distance* models [86] or *additive* models, while DistMult [93] and ComplEx [83] are *multiplicative* models [78], due to the multiplicative score functions used for computing entity-relation-entity triplet likelihood.

The most recent KG embedding models are *ConvE* [16] and *ConvKB* [64]. *ConvE* was the first model using 2D convolutions over embeddings of different embedding dimensions, with the hope of extracting more feature interactions. *ConvKB* replaced 2D convolutions in *ConvE* with 1D convolutions, which constrains the convolutions to be the same embedding dimensions and keeps the translational property of TransE. The other major difference of *ConvE* and *ConvKB* is on the loss functions used in the models. *ConvE* used the cross-entropy loss that could be sped up with 1-N scoring in the decoder, while *ConvKB* used a hinge loss that was computed from positive examples and sampled negative examples. *ConvKB* can be considered as a special case of the proposed *Conv-TransE* that uses filters with width equal to 1. In addition, we take the decoder from *ConvE* because we can easily integrate the encoder of GCN and the decoder of *ConvE* into an end-to-end training framework, while *ConvKB* is not suitable for our approach.

These embedding models achieved good performance for knowledge base completion in terms of efficiency and scalability. However, these approaches only modeled relational triplets, while ignoring a large number of attributes associated with graph nodes, e.g., ages of people or release region of music. Furthermore, these models do not enforce any large-scale connectivity structure in the embedding space, and ignore the knowledge graph structure. The proposed *SACN* handles these two problems in an end-to-end training framework, by using a variant of graph convolutional network (GCN) as the encoder, and a variant of *ConvE* as the decoder.

GCNs were first proposed by Bruna et al. [10], where graph convolutional operations were defined in the Fourier domain. The eigendecomposition of the graph Laplacian caused intense computation. Later, smooth parametric spectral filters [32, 15] were introduced to achieve localization and improve computational efficiency. Recently, Kipf et al. [44] simplified these spectral methods by a first-order approximation with Chebyshev polynomials. It has significantly faster training times and higher predictive accuracy. In addition, many spatial graph convolution approaches [30, 39] define convolutions directly on graphs, which sum up node features over all spatial neighbors using adjacency matrix.

GCN models were mostly criticized for their huge memory requirement to scale to massive graphs. However, one paper [94] developed a data efficient GCN algorithm called PinSage, which combined efficient random walks and graph convolutions to generate embeddings of nodes that incorporated both graph structure as well as node features. The experiments on Pinterest data were the largest application of deep graph embeddings to date with 3 billion nodes and 18 billion edges [94]. This success paves the way for a new generation of web-scale recommender systems based on GCNs. By incorporating a PinSage-like architecture into our Conv-TransE model, we can

take advantage of large and rich graph structures while maintaining computational efficiency.

2.3 Method

In this section, we describe the proposed end-to-end *SACN* as shown in Figure 2.1. The encoder *WGCN* represents entities by aggregating connected entities as specified by the relations in the KB. With node embeddings as the input, the decoder *Conv-TransE* network aims to represent the relations more accurately by recovering the original triplets in the KB. Both encoder and decoder are trained jointly by minimizing the discrepancy (cross-entropy) between the embeddings $e_s + e_r$ and e_o to preserve the translational property $e_s + e_r \approx e_o$. We consider an undirected graph $G = (V, E)$ throughout this section, where V is a set of nodes with $|V| = N$, and $E \subseteq V \times V$ is a set of edges with $|E| = M$.

2.3.1 Weighted Graph Convolutional Layer

The *WGCN* is an extension of classic GCN [44] in the way that it weighs the different types of relations differently when aggregating and the weights are adaptively learned during the training of the network. By this adaptation, the *WGCN* can control the amount of information from neighboring nodes used in aggregation. Roughly speaking, the *WGCN* treats a multi-relational KB graph as multiple single-relational subgraphs where each subgraph entails a specific type of relations. The *WGCN* determines how much weights to give to each subgraph when combining the GCN embeddings for a node.

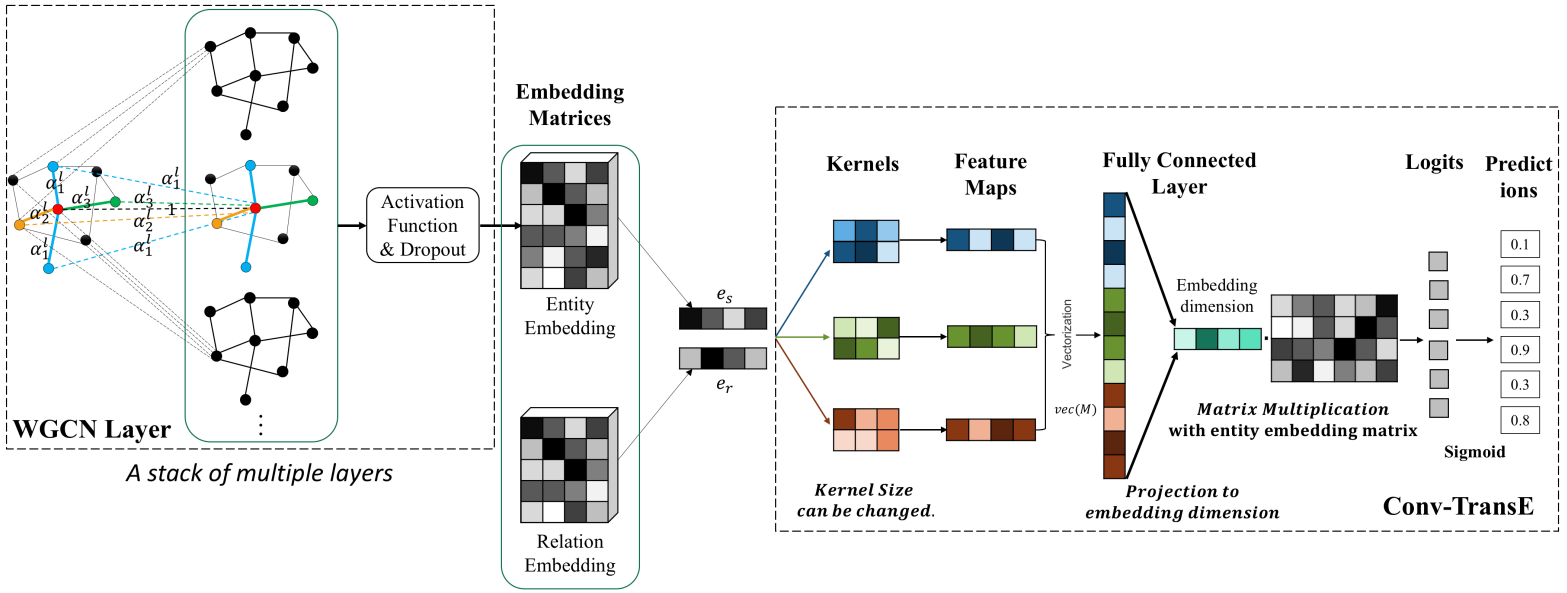


Figure 2.1: An illustration of our end-to-end Structure-Aware Convolutional Networks model. For encoder, a stack of multiple WGCN layers builds an entity/node embedding matrix. For decoder, e_s and e_r are fed into *Conv-TransE*. The output embeddings are vectorized and projected, and matched with all candidate e_o embeddings via inner products. A logistic sigmoid function is used to get the scores.

The l-th *WGCN* layer takes the output vector of length F^l for each node from the previous layer as inputs and generates a new representation comprising F^{l+1} elements. Let h_i^l represents the input (row) vector of the node v_i in the l-th layer, and thus $H^l \in \mathbb{R}^{N \times F^l}$ is the input matrix for this layer. The initial embedding H^1 is randomly drawn from normal (Gaussian) distribution. If there are a total of L layers in the *WGCN*, the output H^{L+1} of the L-th layer is the final embedding. Let the total number of edge types be T in a multi-relational KB graph with E edge set. The interaction strength between two adjacent nodes is determined by their relation type and this strength is specified by a parameter $\{\alpha_t, 1 \leq t \leq T\}$ for each edge type, which is automatically learned in the neural network.

Figure 2.1 illustrates the entire process of *SACN*. In this example, the *WGCN* layers of the network compute the embeddings for the red node in the middle graph. These layers aggregate the embeddings of neighboring entity nodes as specified in the KB relations. Three colors (blue, yellow and green) of the edges indicate three different relation types in the graph. The corresponding three entity nodes are summed up with different weights according to α_t in this layer to obtain the embedding of the red node. The edges with the same color (same relation type) use the same α_t. Each layer has its own set of relation weights α_t^l. Hence, the output of the l-th layer for the node v_i can be written as follows:

$$h_i^{l+1} = \sigma\left(\sum_{j \in \mathbf{N_i}} \alpha_t^l g(h_i^l, h_j^l)\right), \tag{2.1}$$

where $h_j^l \in \mathbb{R}^{F^l}$ is the input for node v_j, and v_j is a node in the neighbor N_i of node v_i. The g function specifies how to incorporate neighboring information. Note that the activation function σ here is applied to every component of its input vector.

Although any function g suitable for a KB embedding can be used in conjunction with the proposed framework, we implement the following g function:

$$g(h_i^l, h_j^l) = h_j^l W^l, \tag{2.2}$$

where $W^l \in \mathbb{R}^{F^l \times F^{l+1}}$ is the connection coefficient matrix and used to linearly transform h_i^l to $h_i^{l+1} \in \mathbb{R}^{F^{l+1}}$.

In Eq. (2.1), the input vectors of all neighboring nodes are summed up but not the node v_i itself, hence self-loops are enforced in the network. For node v_i, the propagation process is defined as:

$$h_i^{l+1} = \sigma(\sum_{j \in \mathbf{N_i}} \alpha_t^l h_j^l W^l + h_i^l W^l). \tag{2.3}$$

The output of the layer l is a node feature matrix: $H^{l+1} \in \mathbb{R}^{N \times F^{l+1}}$, and h_i^{l+1} is the i-th row of H^{l+1}, which represents features of the node v_i in the $(l+1)$-th layer.

The above process can be organized as a matrix multiplication as shown in Figure 2.2 to simultaneously compute embeddings for all nodes through an adjacency matrix. For each relation (edge) type, an adjacency matrix A_t is a binary matrix whose ij-th entry is 1 if an edge connecting v_i and v_j exists or 0 otherwise. The final adjacency matrix is written as follows:

$$A^l = \sum_{t=1}^{T} (\alpha_t^l A_t) + I, \tag{2.4}$$

where I is the identity matrix of size $N \times N$. Basically, the A^l is the weighted sum of the adjacency matrices of subgraphs plus self-connections. In our implementation, we

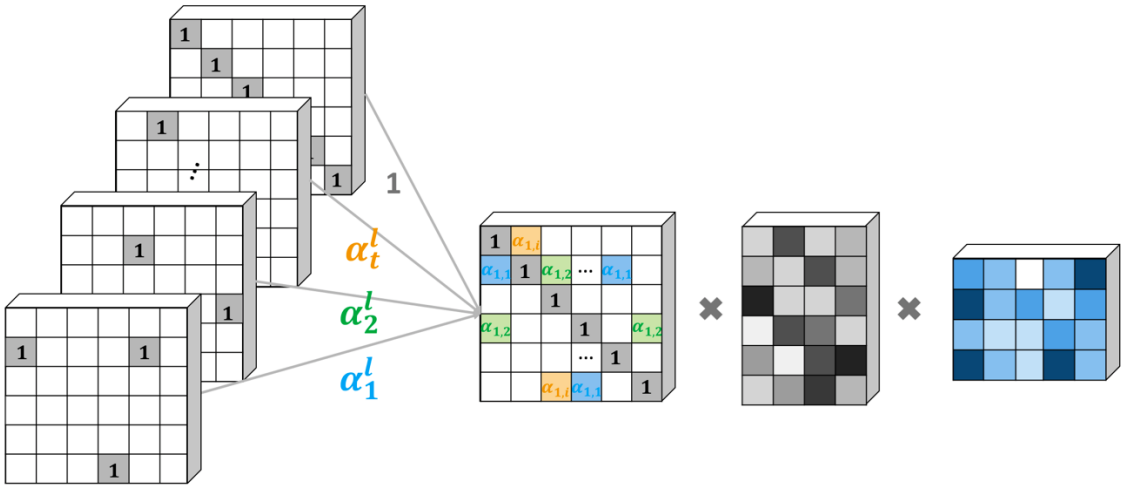


Figure 2.2: A weighted graph convolutional network (*WGCN*) for entity embedding.

consider all first-order neighbors in the linear transformation for each layer as shown in Figure 2.2:

$$H^{l+1} = \sigma(A^l H^l W^l). \tag{2.5}$$

Node Attributes.

In a KB graph, nodes are often associated with several attributes in the form of $(entity, relation, attribute)$. For example, ($s$ = Tom, r = people.person.gender, a = male) is an instance where gender is an attribute associated with a person. If a vector representation is used for node attributes, there would be two potential problems. First, the number of attributes for each node is usually small, and differs from one to another. Hence, the attribute vector would be very sparse. Second, the value of zero in the attribute vectors may have ambiguous meanings: the node does not have the specific attribute, or the node misses the value for this attribute. These zeros would affect the accuracy of the embedding.

In this work, the entity attributes in the knowledge graph are represented by another set of nodes in the network called attribute nodes. Attribute nodes act as the "bridges" to link the related entities. The entity embeddings can be transported over these "bridges" to incorporate the entity's attribute into its embedding. Because these attributes exhibit in triplets, we represent the attributes similarly to the representation of the entity o in relation triplets. In this way, the *WGCN* not only utilizes the graph connectivity structure (relations and relation types), but also leverages the node attributes (a kind of graph structure) effectively. That is why we name our *WGCN* as a structure-aware convolution network.

2.3.2 Conv-TransE

We develop the *Conv-TransE* model as a decoder that is based on *ConvE* but with the translational property of *TransE*: $e_s + e_r \approx e_o$. The key difference of our approach from *ConvE* is that there is no reshaping after stacking e_s and e_r. Filters (or kernels) of size $2 \times k$, $k \in \{1, 2, 3, ...\}$, are used in the convolution. The example in Figure 2.1 uses 2×3 kernels to compute 2D convolutions. We experimented with several of such settings in our empirical study.

Note that in the encoder of *SACN*, the dimension of the relation embedding is commonly chosen to be the same as the dimension of the entity embedding, so in other words, is equal to F^L. Hence, the two embeddings can be stacked. For the decoder, the inputs are two embedding matrices: one $\mathbb{R}^{N \times F^L}$ from *WGCN* for all entity nodes, and the other $\mathbb{R}^{M \times F^L}$ for relation embedding matrix which is trained as well. Because we use a mini-batch stochastic training algorithm, the first step of the decoder performs a look-up operation upon the embedding matrices to retrieve the

input e_s and e_r for the triplets in the mini-batch.

More precisely, given C different kernels where the c-th kernel is parameterized by ω_c, the convolution in the decoder is computed as follows:

$$\begin{aligned} m_c(e_s, e_r, n) = &\sum_{\tau=0}^{K-1} \omega_c(\tau, 0)\hat{e}_s(n+\tau) \\ &+ \omega_c(\tau, 1)\hat{e}_r(n+\tau), \end{aligned} \tag{2.6}$$

where K is the kernel width, n indexes the entries in the output vector and $n \in [0, F^L - 1]$, and the kernel parameters ω_c are trainable. $\hat{e}_s$ and $\hat{e}_r$ are padding version of e_s and e_r respectively. If the dimension s of kernel is odd, the first $\lfloor K/2 \rfloor$ and last $\lfloor K/2 \rfloor$ components are filled with 0. Here $\lfloor value \rfloor$ returns the floor of $value$. Otherwise, the first $\lfloor K/2 \rfloor - 1$ and last $\lfloor K/2 \rfloor$ components are filled with 0. Other components are copied from e_s and e_r directly. As shown in Eq. (2.6) the convolution operation amounts to a sum of e_s and e_r after the one-dimensional convolution. Hence, it preserves the translational property of the embeddings of e_s, e_r. The output forms a vector $M_c(e_s, e_r) = [m_c(e_s, e_r, 0), ..., m_c(e_s, e_r, F^L - 1)]$. Aligning the output vectors from the convolution with all kernels yield a matrix $\mathbf{M}(e_s, e_r) \in \mathbb{R}^{C \times F^L}$.

Finally, the scoring function for the *Conv-TransE* method after the nonlinear convolution is defined as below:

$$\psi(e_s, e_o) = f(vec(\mathbf{M}(e_s, e_r))W)e_o, \tag{2.7}$$

where $W \in \mathbb{R}^{CF^L \times F^L}$ is a matrix for the linear transformation, and f denotes a non-linear function. The feature map matrix is reshaped into a vector $vec(\mathbf{M}) \in \mathbb{R}^{CF^L}$ and projected into a F^L dimensional space using W for linear transformation. Then the

Table 2.1: Scoring function $\psi(e_s, e_o)$. Here $\bar{e_s}$ and $\bar{e_r}$ denote a 2D reshaping of e_s and e_r.

Model	Scoring Function $\psi(e_s, e_o)$
TransE	$\|\|e_s + e_r - e_o\|\|_p$
DistMult	$\langle \mathrm{e}_s, e_r, e_o \rangle$
ComplEx	$\langle \mathrm{e}_s, e_r, e_o \rangle$
ConvE	$f(vec(f(concat(\bar{e_s}, \bar{e_r}) * \omega))W)e_o$
ConvKB	$concat(g([e_s, e_r, e_o] * \omega))\beta$
SACN	$f(vec(\mathbf{M}(e_s, e_r))W)e_o$

calculated embedding is matched to e_o by an appropriate distance metric. During the training in our experiments, we apply the logistic sigmoid function σ to the scoring:

$$p(e_s, e_r, e_o) = \sigma(\psi(e_s, e_o)). \tag{2.8}$$

In Table 2.1, we summarize the scoring functions used by several state of the art models. The vector e_s and e_o are the subject and object embedding respectively, e_r is the relation embedding, "concat" means concatenates the inputs, and "*" denotes the convolution operator.

In summary, the proposed *SACN* model takes advantage of knowledge graph node connectivity, node attributes and relation types. The learnable weights in *WGCN* help to collect adaptive amount of information from neighboring graph nodes. The entity attributes are added as additional nodes in the network and are easily integrated into the *WGCN*. *Conv-TransE* keeps the translational property between entities and relations to learn node embeddings for the link prediction. We also emphasize that our *SACN* has significant improvements over *ConvE* with or without the use of node attributes.

Table 2.2: Statistics of datasets.

Dataset	FB15k-237	WN18RR	FB15k-237-Attr
Entities	14,541	40,943	14,744
Relations	237	11	484
Train Edges	272,115	86,835	350,449
Val. Edges	17,535	3,034	17,535
Test Edges	20,466	3,134	20,466
Attributes Triples	—	—	78,334
Attributes	—	—	203

2.4 Experiments

2.4.1 Benchmark Datasets

Three benchmark datasets (FB15k-237, WN18RR and FB15k-237-Attr) in Table 2.2 are utilized in this study to evaluate the performance of link prediction.

FB15k-237. The FB15k-237 [82] dataset contains knowledge base relation triples and textual mentions of Freebase entity pairs, as used in the work published in [82]. The knowledge base triples are a subset of the FB15K [7], originally derived from Freebase. The inverse relations are removed in FB15k-237.

WN18RR. WN18RR [16] is created from WN18 [7], which is a subset of WordNet. WN18 consists of 18 relations and 40,943 entities. However, many text triples are obtained by inverting triples from the training set. Thus WN18RR dataset [16] is created to ensure that the evaluation dataset does not have inverse relation test leakage. In summary, WN18RR dataset contains 93,003 triples with 40,943 entities and 11 relation types.

2.4.2 Data Construction

Most of the previous methods only model the entities and relations, and ignore the abundant entity attributes. Our method can easily model a large number of entity attribute triples. In order to prove the efficiency, we extract the attribute triples from the FB24k [53] dataset to build the evaluation dataset called FB15k-237-Attr.

FB24k. FB24k [53] is built based on Freebase dataset. FB24k only selects the entities and relations which constitute at least 30 triples. The number of entities is 23,634, and the number of relations is 673. In addition, the reversed relations are removed from the original dataset. In the FB24k datasets, the attribute triples are provided. FB24k contains 207,151 attribute triples and 314 attributes.

FB15k-237-Attr. We extract the attribute triples of entities in FB15k-237 from FB24k. During the mapping, there are 7,589 nodes from the original 14,541 entities which have the node attributes. Finally, we extract 78,334 attribute triples from FB24k. These triples include 203 attributes and 247 relations. Based on these triples, we create the "FB15k-237-Attr" dataset, which includes 14,541 entity nodes, 203 attribute nodes, 484 relation types. All the 78,334 attribute triples are combined with the training set of FB15k-237.

2.4.3 Experimental Setup

The hyperparameters in our *Conv-TransE* and *SACN* models are determined by a grid search during the training. We manually specify the hyperparameter ranges: learning rate $\{0.01, 0.005, 0.003, 0.001\}$, dropout rate $\{0.0, 0.1, 0.2, 0.3, 0.4, 0.5\}$, node embedding size $\{100, 200, 300\}$, number of kernels $\{50, 100, 200, 300\}$, and kernel size

$\{2 \times 1, 2 \times 3, 2 \times 5\}$.

Here all the models use the *WGCN* with two layers. For different datasets, we have found that the following settings work well: for FB15k-237, set the dropout to 0.2, number of kernels to 100, learning rate to 0.003 and embedding size to 200 for *SACN*; for WN18RR dataset, set dropout to 0.2, number of kernels to 300, learning rate to 0.003, and embedding size to 200 for *SACN*. When using the *Conv-TransE*-alone model, these settings still work well.

Each dataset is split into three sets for: training, validation and testing, which is same with the setting of the original *ConvE*. We use the adaptive moment (Adam) algorithm [41] for training the model. Our models are implemented in PyTorch and run on NVIDIA Tesla P40 Graphics Processing Units.

2.4.4 Results

Evaluation Protocol

Our experiments use the proportion of correct entities ranked in top 1,3 and 10 (Hits@1, Hits@3, Hits@10) and the mean reciprocal rank (MRR) as the metrics. In addition, since some corrupted triples exist in the knowledge graphs, we use the filtered setting [7], i.e. we filter out all valid triples before ranking.

Link Prediction

Our results on the standard FB15k-237, WN18RR and FB15k-237-Attr are shown in Table 2.3. Table 2.3 reports Hits@10, Hits@3, Hits@1 and MRR results of four different baseline models and two models of our approach on three knowledge graphs

Table 2.3: Link prediction for FB15k-237, WN18RR and FB15k-237-Attr datasets. $SACN^1$ uses the FB15k-237 dataset and $SACN^2$ uses FB15k-237-Attr dataset.

	FB15k-237				**WN18RR**			
	Hits				Hits			
Model	@10	@3	@1	MRR	@10	@3	@1	MRR
DistMult [93]	0.42	0.26	0.16	0.24	0.49	0.44	0.39	0.43
ComplEx [83]	0.43	0.28	0.16	0.25	0.51	0.46	0.41	0.44
R-GCN [72]	0.42	0.26	0.15	0.25	—	—	—	—
ConvE [16]	0.49	0.35	0.24	0.32	0.48	0.43	0.39	0.46
Conv-TransE	0.51	0.37	0.24	0.33	0.52	0.47	0.43	0.46
SACN[1]	0.54	0.39	0.26	0.35	**0.54**	**0.48**	**0.43**	**0.47**
SACN[2]	**0.55**	**0.40**	**0.27**	**0.36**	—	—	—	—

datasets. The FB15k-237-Attr dataset is used to prove the efficiency of node attributes. So we run our SACN in FB15k-237-Attr to do the comparison with SACN using FB15k-237.

We first compare our *Conv-TransE* model with the four baseline models. *ConvE* has the best performance comparing all baselines. In FB15k-237 dataset, our *Conv-TransE* model improves upon ConvE's Hits@10 by a margin of 4.1% , and upon ConvE's Hits@3 by a margin of 5.7% for the test. In WN18RR dataset, *Conv-TransE* improves upon ConvE's Hits@10 by a margin of 8.3% , and upon ConvE's Hits@3 by a margin of 9.3% for the test. For these results, we conclude that *Conv-TransE* using neural network keeps the translational characteristic between entities and relations and achieve better performance.

Secondly, the structure information is added into our *SACN* model. In Table 2.3, *SACN* also gets the best performances in the test dataset comparing all baseline methods. In FB15k-237, comparing *ConvE*, our *SACN* model improves Hits@10 value by a margin of 10.2%, Hits@3 value by a margin of 11.4%, Hits@1 value by a margin

of 8.3% and MRR value by a margin of 9.4% for the test. In WN18RR dataset, comparing *ConvE*, our *SACN* model improves Hits@10 value by a margin of 12.5%, Hits@3 value by a margin of 11.6%, Hits@1 value by a margin of 10.3% and MRR value by a margin of 2.2% for the test. So our method has significant improvements over *ConvE* without attributes.

Thirdly, we add node attributes into our *SACN* model, i.e. we use the FB15k-237-Attr to train *SACN*. Note that *SACN* has significant improvements over ConvE without attributes. Adding attributes improves performance again. Our model using attributes improves upon ConvE's Hits@10 by a margin of 12.2% , Hits@3 by a margin of 14.3%, Hits@1 by a margin of 12.5% and MRR by a margin of 12.5%. In addition, our *SACN* using attributes improved Hits@10 by a margin of 1.9% , Hits@3 by a margin of 2.6%, Hits@1 by a margin of 3.8% and MRR by a margin of 2.9% comparing with *SACN* without attributes.

In order to better compare with *ConvE*, we also use the attributes into *ConvE*. Here the attributes will be treated as the entity triplets. Following the official *ConvE* code with default setting, the test result in FB15k-237-Attr was: 0.46 (Hits@10), 0.33 (Hits@3), 0.22 (Hits@1) and 0.30 (MRR). Comparing to the performance without the attributes, adding the attributes into the *ConvE* didn't improve performance.

Convergence Analysis

Figure 2.3 shows the convergence of the three models. We can see that the *SACN* (the red line) is always better than *Conv-TransE* (the yellow line) after several epochs. And the performance of *SACN* keeps increasing after around 120 epochs. However, the *Conv-TransE* has achieved the best performance after around 120 epochs. The

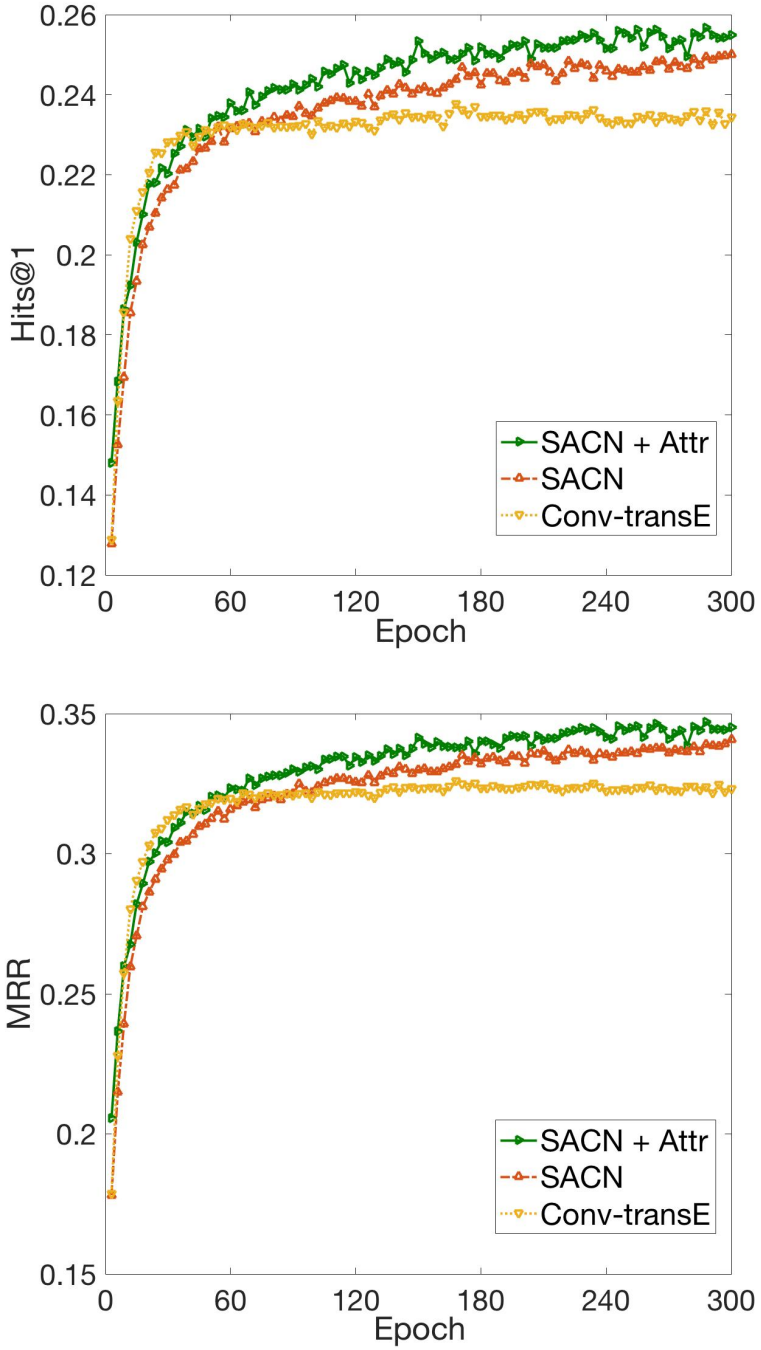


Figure 2.3: The convergence study of *SACN*, *Conv-TransE* models in FB15k-237 and *SACN* in FB15k-237-Attr (*SACN + Attr*) using the validation set. Due to the page limitation, only the results of Hits@1 and MRR are reported here.

Table 2.4: Kernel size analysis for FB15k-237 and FB15k-237-Attr datasets. "*SACN+Attr*" means the *SACN* using FB15k-237-Attr dataset.

Model	Kernel Size	**FB15k-237**			
		Hits			
		@10	@3	@1	MRR
Conv-TransE	2×1	0.504	0.357	0.234	0.324
Conv-TransE	2×3	0.513	0.365	0.240	0.331
Conv-TransE	2×5	0.512	0.361	0.239	0.329
SACN	2×1	0.527	0.379	0.255	0.345
SACN	2×3	0.536	0.384	0.260	0.351
SACN	2×5	0.536	0.385	0.261	0.352
SACN+Attr	2×1	0.535	0.384	0.260	0.351
SACN+Attr	2×3	0.543	0.394	0.268	0.360
SACN+Attr	2×5	**0.547**	**0.396**	**0.268**	**0.360**

gap between these two models proves the usefulness of structural information. When using the FB15k-237-Attr dataset, the performance of "*SACN + Attr*" is better than "*SACN*" model.

Kernel Size Analysis

In Table 2.4, different kernel sizes are examined in our models. The kernel of "2×1" means the knowledge or information translating between one attribute of entity vector and the corresponding attribute of relation vector. If we increase the kernel size to "$2 \times k$" where $k = \{3, 5\}$, the information is translated between a combination of s attributes in entity vector and a combination of k attributes in relation vector. The larger view to collect attribute information can help to increase the performance as shown in Table 2.4. However, the optimal kernel size may be dependent on tasks.

Table 2.5: Node indegree study using FB15k-237 dataset.

	Conv-TransE		**SACN**	
	Average Hits		Average Hits	
Indegree Scope	@10	@3	@10	@3
[0,100]	0.192	0.125	0.195	0.134
[100,200]	0.441	0.245	0.441	0.253
[200,300]	0.696	0.446	0.705	0.429
[300,400]	0.829	0.558	0.806	0.577
[400,500]	0.894	0.661	0.868	0.663
[500,1000]	0.918	0.767	0.891	0.695
[1000, maximum]	0.992	0.941	0.981	0.922

Node Indegree Analysis

The indegree of the node in knowledge graph is the number of edges connected to the node. The node with larger degree means it have more neighboring nodes, and this kind of nodes can receive more information from neighboring nodes than other nodes with smaller degree. As shown in Table 2.5, we present the results for different sets of nodes with different indegree scopes. The average Hits@10 and Hits@3 scores are calculated. Along the increasing of indegree scope, the average value of Hits@10 and Hits@3 will be increased. First for a node with small indegree, it benefits from aggregation of neighbor information from the WGCN layers of *SACN*. Its embedding can be estimated robustly. Second for a node with high indegree, it means that a lot more information is aggregated through GCN, and the estimation of its embedding is substantially smoothed among neighbors. Thus the embedding learned from *SACN* is worse than that from *Conv-TransE*. One solution to this problem would be neighbor selection as in [94].

Chapter 3

Structure-Aware Learning on Small-Scale Molecular Graphs

3.1 Motivation

In recent years, there has been a growing interest in incorporating deep learning approaches into chemoinformatics studies [60, 55, 19], such as quantitative structure-activity relationship (QSAR) prediction, library diversity analysis, and numerical representations of molecules. Most QSAR studies have employed some hand-crafted molecular descriptors or fingerprints [24] or directly SMILE strings. To extract better representation using the chemical structure and bonding, a chemical compound can be expressed as a hydrogen-depleted molecular graph whose nodes correspond to (non-hydrogen) atoms while edges with discrete attributes represent chemical bonds. Recently, methods emerge to derive representations using molecular graphs directly with deep graph convolutional networks [18, 89, 23, 38]. However, current GCN

methods have many limitations to handle the molecular graphs.

The key limitation is that existing approaches ignore the general consistency of the bond energies (enthalpies) and bond lengths (interatomic distances) in various molecules [52, 11]. The bond dissociation energy, also known as bond enthalpy, is defined as the standard enthalpy change when a bond is cleaved by homolysis. So the bond energy is the amount of energy required to break a bond. The equilibrium bond length is the average distance between the nuclei of two bonded atoms in a molecule. The higher the bond energy, the "stronger" the bond is between the two atoms, and the length of the bond is smaller. The bond energies and bond lengths between two specific atoms are generally consistent across different molecules. The tables about bond lengths (interatomic distances) [69] and bond energies [34] have been used in some standard handbooks, which is used and consistent in all molecules. For example, the bond length of the atom pair Carbon-Nitrogen (C-N) is 147 pm and the bond length of Carbon-Carbon (C-C) is 154 pm. In addition, the bond length is also affected by other factors, such as bond order. For example, the bond length of C=C is 134 pm and the bond length of C≡C is 120 pm. Overall, this means the edges in different molecular graphs present some consistency, which we call "edge consistency". That means all edges with same type should provide the same functionality when passing information [23] between atoms. Since the edges have this property, we aim to design a new GCN with a constraint to keep this edge consistency. By designing a new attention mechanism, our proposed model can pass messages along similar edges with similar attention weights.

In this work, we propose a consistent Edge-Aware multi-view spectral GCN approach (*EAGCN*) to enhance the molecular graph property prediction (also called QSAR prediction) by learning more accurate molecular representations. Our model

Table 3.1: Edge attributes commonly used in molecular graphs.

Attribute	Description
`Atom Pair Type`	Defined by the type of the atoms that a bond connects (e.g., C-C, C-O).
`Bond Order`	Bond order (single bond, aromatic bond, double bond and triple bond).
`Aromaticity`	Is aromatic.
`Conjugation`	Is conjugated.
`Ring Status`	Is in a ring.

enforces an edge consistency constraint that similar edges should have similar weights across all molecular graphs in a dataset. In addition, our model provides a new way to take advantage of the discrete edge attributes by creating multi-view graphs.

Firstly, to design an edge consistency constraint, we explore the attributes of edges in molecular graphs. As illustrated in Table 3.1, these discrete attributes are important to describe an edge or bond [14]. Each discrete value of an edge attribute is considered as an edge type. For instance, for one edge, we notice that whether the bond is formed between the Carbon and Oxygen atoms (i.e., the C-O bond) and whether the bond is aromatic and so on. These attributes of edges are highly related to the bond lengths and energies. In the molecular graph, the edges with the same attribute values have similar bond lengths and energies.

Based on these edge attributes, we present a consistent edge mapping (CEM) mechanism to learn the consistent attention weights of edges in order to receive different amounts of information from neighbors and enforce the edges with the same type to pass the same amount of the information. The learned real-valued attention matrix assigns a non-negative weight to an edge, and assigns 0 when there is no edge between two nodes. Importantly, an effective way is provided to enforce consistent

attention weights by building edge mapping dictionaries shown in section 3.4.1. The same weight from the edge dictionary will be provided if two edges have the same edge attribute value. The weights of these edge dictionaries, which are the parameters in the neural networks, are shared for different molecules, which promotes *EAGCN* to learn the inherently invariant properties in the graphs. In graph theory, graph invariant is defined as a property preserved under any possible isomorphism of a graph, which includes the order invariant, permutation invariant and pair order invariant [38, 40].

Secondly, since we have multiple attributes for each edge as shown in Table 3.1 [14], we model a molecular graph as multi-view graphs for taking advantage of the edge attributes, where each attribute including a group of discrete values is defined as one view. Hence a molecular graph can be decomposed into multiple graphs from different views according to each specific edge attribute. The current GCN methods predefine a single fixed adjacency matrix A for a graph, which cannot catch the different edge attributes. In our approach, the discrete values of one attribute are used to create one edge mapping dictionary including attention weights for this corresponding view. Notice that each view will keep edge (relation) consistency by using a consistent attention mechanism.

Furthermore, a new spectral filter is derived from the first order Chebyshev approximation which relies on two coefficients - the 0^{th} order and 1^{st} order coefficients (θ_0 and θ_1). These two coefficients are enforced to be opposed to each other ($\theta_0 = -\theta_1$) in all early spectral graph convolution [44]. We relax this requirement, resulting in the more general and effective parameterization of the GCN network. In addition, to create a fingerprint for a molecule, learning the graph representation for the entire molecule is also important for us. We apply two ways including simple sum and

differential pooling [95] to integrate node embeddings into a graph representation. Finally, *EAGCN* provides an alternative fingerprints for chemical compounds other than hand-crafted ones, and prove to be useful in predicting the molecular graph properties in the experiments.

The major contributions are summarized as follows:

1. We propose a consistent edge-aware multi-view spectral GCN model named *EAGCN* that preserves the edge (bond) consistency in molecular graphs. The proposed CEM mechanism in *EAGCN* learns the consistent edge attention weights cross molecules by building global edge dictionaries.

2. Multi-view graphs have been employed in our *EAGCN* model by taking advantage of the discrete edge attributes. Here each edge attribute is characterized in a view of the molecular graph to explore the multiple types of edges in molecular graphs.

3. A new spectral filter with fine approximation is designed for the spectral graph convolutions. This spectral convolution operation allows more flexible and general network parameterization.

3.2 Related Work

3.2.1 Spectral Graph Convolutions

The graph convolution operations are first defined by Bruna et al. [10] in the Fourier domain. Since the eigendecomposition of the graph Laplacian is needed, it involves intense computation. To tackle this issue, smooth parametric spectral filters [32] were

introduced to achieve localization in the spatial domain and computational efficiency. In particular, Chebyshev polynomials [15] and Cayley polynomials [49] have been utilized in these convolutional architectures to efficiently produce localized filters. Recently, Kipf et al. [44] simplified these spectral methods by a first-order approximation of the Chebyshev polynomials. Their derivation finally leads to localization of one-step neighbors and achieves state-of-the-art performance.

However, the spectral filters learned by the above methods depend on the Laplacian eigenbasis which is linked to a fixed graph structure. Thus these models can only be trained on a single graph, and cannot be directly used to model a set of graphs with different structures. Recently graph attention networks [85] have been proposed to deal with arbitrary graphs without knowing the entire graph structures. The attention mechanisms allow the model to deal with inputs of varying sizes. This attention-based architecture assigns different weights to different nodes within a neighborhood while dealing with various sized neighborhoods. It determines the attention weight between any two nodes by calculating the similarity of the two node feature vectors. It has not considered the case where edge attentions can be directly learned. Inspired by this work, we design a new edge attention mechanism in this dissertation to understand which types of edges (bonds) are important for the target molecular property instead of using node similarities.

3.2.2 Non-spectral Graph Convolutions

The spatial graph convolution approaches [18, 3, 30] define convolutions directly on graph, which sum up node features over all spatially close neighbors by multiplying an adjacency matrix which is pre-defined rather than learnable. From another per-

spective, the spatial graph convolution methods are summarized as a message passing algorithm and aggregation procedure [23]. Another approach [67] enables the traditional CNNs to be applied to graph inputs directly by selecting fixed-size neighbors and normalizing these nodes. For a large-scale graph, Hamilton [30] introduces the task of inductive node classification, where the goal is to classify nodes that have not been seen during training. This approach samples a fixed-size of neighbors for each node and achieves state-of-the-art performance on several datasets. For multiple small-scale graphs, Duvenaud et al. [18] first present a CNN to operate directly on raw molecular graphs, that creates prediction pipelines whose inputs are molecular graphs of arbitrary size.

3.2.3 Convolutions on Molecular Graphs

Neural networks and GCNs have been applied to studies such as protein interface prediction [19], molecular representation and prediction [18, 23, 38, 14, 50]. Duvenaud et al. [18] present a CNN that operates directly on raw molecular graphs and generalize standard molecular feature extraction methods based on circular fingerprints (ECFP) [71]. Based on an autoencoder model, the method in [25] converts discrete representations (e.g., ECFP) of molecules to a multidimensional continuous one. Another work [38] attempts to use the different attributes of chemical bonds, graph edges are associated with attributes such as atom-pair type or the bond order and modeled via tensor computation. The resultant graph network utilizes properties of both nodes (atoms) and edges (bonds). The method in [14] learns atom embeddings which are then concatenated by the related bond features to form atom-bond feature vectors. In these works, node features and bond attributes are treated equally and used the

way in the neighborhood aggregation. However, different types of bonds play very different roles, it is more natural that they receive different attentions in relation to a target property. In addition, in order to handle graphs of varying size and connectivity, Simonovsky et al. [79] propose the edge convolutional network (ECC) for point cloud classification. Their model defines several node feature filters based on the edge label.

3.3 Problem Formulation

Let $G = (V, E)$ be an undirected graph where V is a set of n nodes, $E \subseteq V \times V$ is a set of n_E edges. The node features from a graph are represented by a feature matrix $X \in \mathbb{R}^{n \times f_1}$ which is the input of layer 1 so that we get $X^1 = X$. The edge features from a graph are represented by $\mathbb{Z} = \{Z_1, ..., Z_k\}$, here k is the number of edge features as shown in Table 3.1 and $Z_i \in \mathbb{Z}$ has multiple discrete values. The label for each input graph is the molecular graph property $y \in Y$. Hence the task is to predict the target molecular properties for all input graphs.

In traditional GCNs [44], the connectivity of the graph is summarized into a binary adjacency matrix A_{binary} where an entry has a value of 1 if there is an edge connecting the related two nodes; or otherwise has a value of 0. Based on A_{binary}, the spectral convolution on graph is defined as the multiplication of a signal $x \in X$ with a filter g_θ parameterized by θ from matrix coefficients Θ:

$$g_\theta \star x = U g_\theta U^T x, \tag{3.1}$$

where U is the matrix of eigenvectors of the normalized graph Laplacian $L = I_N -$

$D^{1/2}A_{binary}D^{1/2}$, where I is the identity matrix and D is the degree matrix of the fixed binary adjacency matrix A_{binary}.

However, when modeling molecular graphs, we need to pay attention to two properties. Firstly, some bonds can be more important than others for a specific structure-activity relationship, which cannot be implemented by a binary adjacency matrix which amounts to equally sum up the information of all neighbors. Naturally different attention weights should be given to different neighbors when aggregating node embeddings, in order to predict a graph property. For example, in the aromaticity view, the weights for aromatic bonds in any molecules can be closer than those assigned to non-aromatic bonds. Secondly, various molecular graphs have the consistency of the bond energies (enthalpies) and bond lengths (interatomic distances), called edge consistency, which has been illustrated in the motivation. The bonds with the same bond type should be assigned one same attention weight.

To the best of our knowledge, there has been no attempt to design a special attention mechanism for modeling the edge consistency in molecules. In the latest research works, attention-based GCNs have been studied [85, 48] which learn a dynamic and adaptive aggregation of the neighborhood. These methods such as GAT [48] determine the weight for an edge based on the similarity of the two end nodes calculated each time when the node embeddings are updated. However, all approaches ignore the edge consistency, which is an important property of molecules. In addition, most of the existing methods [30, 10, 85, 48] including GAT are designed for one large graph. It increases the difficulty of designing a consistent edge constrained attention method when the inputs are multiple molecular graphs with different sizes and shapes. All edges with same type across all molecular graphs should share one same attention weight.

In this dissertation, we propose a Consistent Edge Mapping (CEM) mechanism, which not only assigns different attention weights to different neighbors when aggregating node embedding, but also enforces a constraint of edge consistency that all edges of same interaction type should be assigned one same attention weight cross different input molecular graphs. This CEM mechanism takes advantage of the two properties in molecules.

In addition, we model a molecular graph as multi-view graphs for taking advantage of the edge attributes. Each edge in the molecular graph has multiple edge attributes as shown in Table 3.1, where each attribute has several discrete values. Each discrete value from one edge attribute represents one edge type or one atom-to-atom interaction. Hence, for each edge attribute, we can get a group of atom-to-atom interactions or edge types. In this approach, each attribute including a group of edge types is defined as one view. For each view, our CEM function C defined in the following section will be able to build the consistent edge constraint by assigning attention weights based on this group of edge types. Notice that this group of edge types will be shared across all molecules.

After that, each view is represented as a graph with a consistent edge attention matrix where edge weights are specified according to edge types. Since we have multiple discrete edge attributes, a molecular graph can be decomposed into multiple graphs with a group of consistent edge attention matrices $\mathbb{A} = \{A_0, A_1, ..., A_k\}$. Based on function C and multiple edge attributes $\mathbb{Z}$, a novel edge consistent constraint based multi-view spectral graph convolution is proposed to collectively aggregate information from graph structure.

Firstly, let's consider the i-th edge attribute (i-th view) $Z_i \in \mathbb{R}^{d_i \times 1}$ where d_i is the number of discrete values of attribute i. The objective is to learn a function $F_{\Theta,M}$ and

a function P_W minimizing the empirical loss $\mathbb{L}$ with an edge consistency constraint:

$$\min_{\substack{\Theta^1,...,\Theta^L,W \\ M^1,...,M^L}} \mathbb{L}(P(F^L \circ F^{L-1} \circ ... \circ F^1(X^1; A_i^1, \Theta^1); W), y)$$
$$\text{subject to } A_i^l = C(A_{binary}, Z_i; M^l), \forall 1 \leq l \leq L, X^1 = X,$$

where $C(A_{binary}, Z_i; M^l)$ is the consistent edge mapping (CEM) function, M^l includes all parameters of C in layer l. The function P includes the graph representation layer, fully connected layer and activation function where W is all parameters of P. F is the graph convolutional network, $\circ$ is the composite function where $F^{l+1} \circ F^l := F^{l+1}(F^l(X^l; A^l, \Theta^l); A^{l+1}, \Theta^{l+1}))$ and Θ^l includes all parameters of F in layer l.

Secondly, we extend our model to a multi-view neural network by using the multiple attributes $\mathbb{Z} = \{Z_1, ..., Z_k\}$. In multi-view model, our the objective function can be written as:

$$\min_{\substack{\Theta^1,...,\Theta^L,W \\ M^1,...,M^L}} \mathbb{L}(P(F^L \circ F^{L-1} \circ ... \circ F^1(X^1; \mathbb{A}^1, \Theta^1); W), y)$$
$$\text{subject to } \mathbb{A}^l = \{A_0^l, A_1^l, ..., A_k^l\}, A_i^l = C(A_{binary}, Z_i; M_i^l), \forall 1 \leq l \leq L, X^1 = X,$$

where the parameters M^l and Θ^l can be defined as $M^l = \{M_1^1, ..., M_k^l\}$ and $\Theta^l = \{\Theta_1^1, ..., \Theta_k^l\}$ and F is the proposed multi-view graph convolutional network layer.

In summary, we define the molecular property prediction task as an edge consistency constraint based graph learning problem, which reserves the general consistency of the bond energies and bond lengths in various molecules.

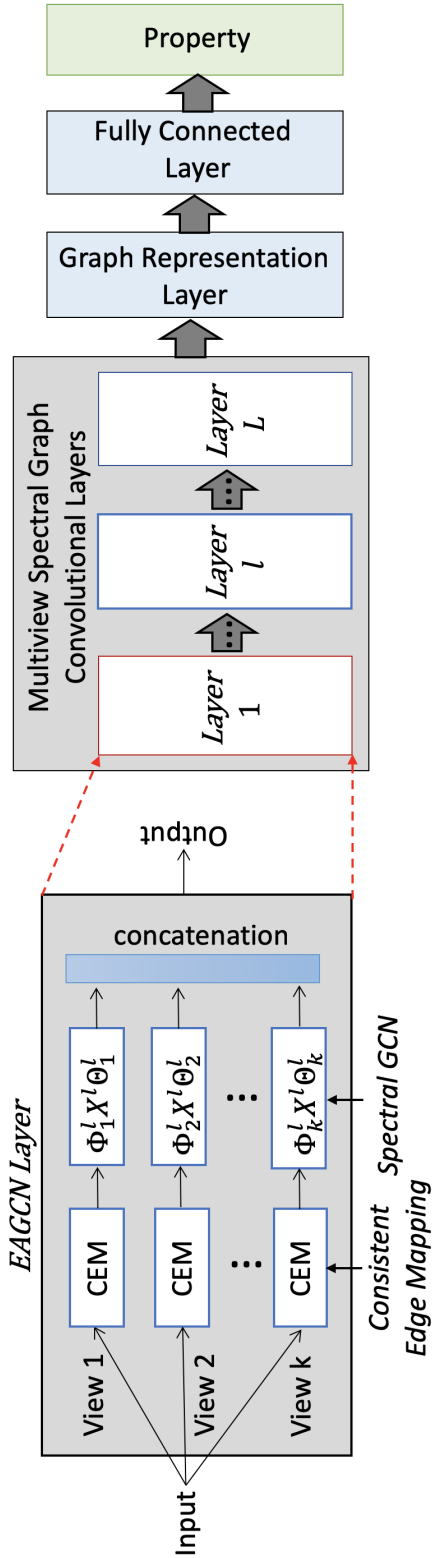


Figure 3.1: The architecture of $EAGCN$ model.

3.4 The EAGCN Model

The architecture of our proposed *EAGCN* model is shown in Figure 3.1. The *EAGCN* begins with multi-view spectral graph convolutional layers where each layer has two steps: Consistent Edge Mapping (CEM) and Spectral GCN. The output of multi-view spectral graph convolutional layers is the node embeddings. Then a graph representation layer combines all node embedding to get the graph embedding. Here we apply two ways including simple sum and differential pooling [95] to integrate node embeddings into a graph representation. Then the property values of the molecular graphs are predicted through the fully connected layer with the activation function.

3.4.1 Consistent Edge Mapping (CEM)

The CEM mechanism is proposed to enforce the edge consistency constraint. It generates a learnable consistent edge attention matrix A_i^l using the discrete values of the edge attribute i (view i) in layer l. Here we consider two edges have same edge type if two edges in one molecule or different molecules have same edge attribute value. The set of discrete edge attribute values (edge types) is the edge priori information used to limit the space to be searched. More importantly, since this set of edge types is shared for any molecules, we have the edge consistency by assigning the same weight to all edges with same type across molecular graphs. In this method, this consistency of assigned edge attention weights for all input graphs is called "edge consistency constraint".

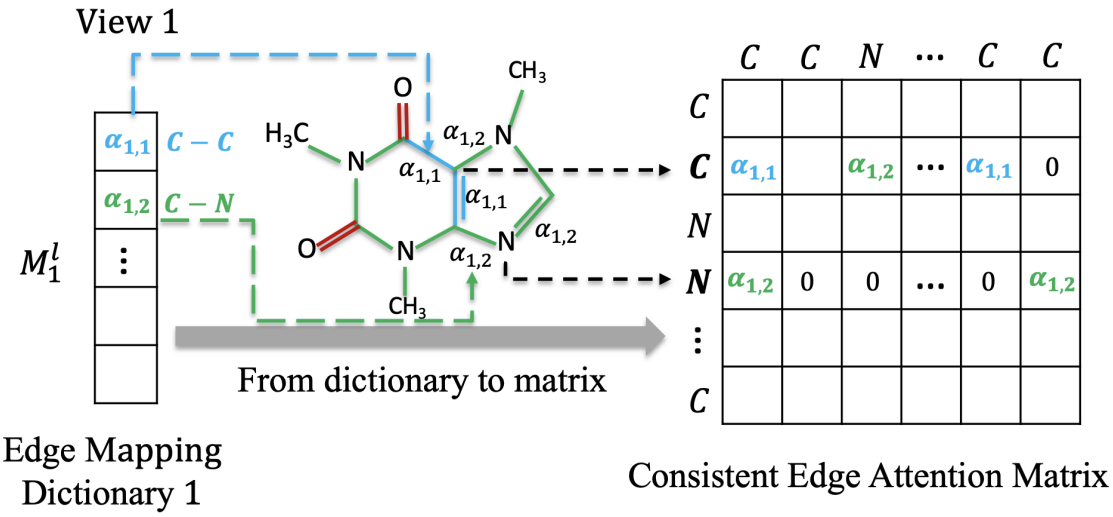


Figure 3.2: Example of the edge mapping dictionary for view 1 in layer l.

Creating Edge Mapping Dictionaries

To better understand the importance of each edge type to the target graph property, *EAGCN* learns the attention weights for graph edges at each neural network layer. However, it is difficult to assign weights with edge consistency for multiple graphs with different sizes and shapes. Here edge consistency means the same type edges in the same and different graphs need to be assigned one same learnable weight in neural network.

In our approach, for each view i in layer l, we build an edge mapping dictionary M_i^l to solve the consistent edge assignment problem. For view i, if we have d_i discrete values for edge attribute i, edge mapping dictionary M_i^l is created with d_i learnable neural network parameters. Here one discrete value represents one edge type, so there is one corresponding parameter for this edge type in edge mapping dictionary. Since these neural network parameters are shared for all molecules during the learning process, we call them "consistent edge attention weights". For example, as shown in

Figure 3.2, all edges will search the edge mapping dictionary to get the attention weight in one view. If the type of two edges is same, the attention weights for them will be the same as well. In Figure 3.2, we can see that all C-C edges will select the same $\alpha_{1,1}$ as the attention weight from M_1^l in layer l.

The attention weights in these dictionaries will be learned by our *EAGCN* model. We assert two points for the dictionaries: Firstly, the edge mapping dictionaries are defined for one dataset. Based on its discrete values of edge attributes, each molecule is coded as weighted adjacency matrices of multiple views using these edge mapping dictionaries; Secondly, the dictionaries for edge attributes are not only shared in one graph, but also used for all graphs in the dataset. Then a weighted adjacency matrix called a consistent edge attention matrix for each view and each layer is constructed according to the edge mapping dictionary in the following subsection.

Consistent Edge Attention Matrix

In layer l, we first create a binary one-hot encoding vector [1] $b_i(e) \in \mathbb{R}^{d_i}$ for each edge $e \in E$ in the view i if there exists edge connecting two neighbor nodes, otherwise it will be a zero vector. Based on the vector $b_i(e)$, we will obtain a specific weight $\alpha_{i,j}^l$ for edge e by looking up dictionary:

$$\alpha_{i,j}^l = b_i(e)^T M_i^l, \tag{3.2}$$

here j denotes the j-th element in the dictionary M_i^l, i means the view i. Notice that the $\alpha_{i,j}^l$ will be zero if $b_i(e)$ is a zero vector. As shown in Figure 3.3, all vectors form

[1] One-hot encoding vector is a group of values among which the legal combinations of values are only those with a single 1 and all the others 0.

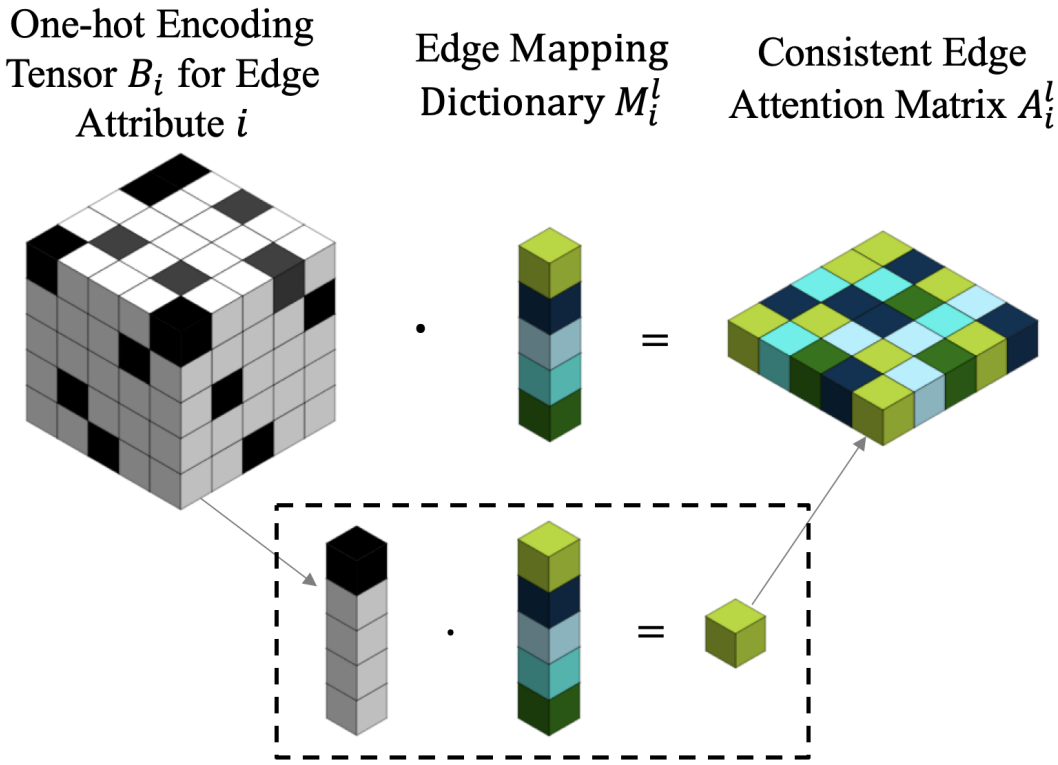


Figure 3.3: The process of consistent edge mapping.

the binary tensor B_i for view i. By looking up all existing edges using the binary tensor, we get a newly generated consistent edge attention matrix A_i^l:

$$A_i^l = B_i \cdot M_i^l, \tag{3.3}$$

here "$\cdot$" is the lookup operation on the dictionary as shown in Figure 3.3. Since the mapping dictionaries which are global parameters are shared for all graphs, so we call this step the consistent edge attention mapping operation. The global edge weights across all graphs preserve the consistent property of the relationship. In the experiment, we implement this mapping process using 2D convolution. We consider the one-hot encoding tensor is an image with multiple channels. Here the channel size is the dimension d_i of the edge mapping dictionary. Since the edge mapping dictionary which contains the d_i parameters of neural network is $1 \times 1 \times d_i$, so the kernel size we used is 1×1.

3.4.2 Multi-view Spectral GCN

The previous section introduces the CEM to transform the edge priori information of graph signals to the edge consistency constraint. Based on this constraint, each molecular graph in layer l has been converted to a constrained graph signal, which consists of consistent edge adjacency matrices $\mathbb{A} = \{A_1^l, ..., A_k^l\}$ and feature matrix X. In this section, we propose a multi-view spectral graph convolution operation with a new spectral filter for the graph signal. Firstly, our proposed multi-view GCN with the layer-wise propagation rule is presented in subsection 3.4.2. Secondly, subsection 3.4.2 illustrates this propagation rule is motivated via a proposed new

spectral filter with the more reasonable parameterization of first order approximation on the constrained graph signals.

Layer-wise propagation rule of Proposed Graph Convolution

In the proposed approach, each molecular graph has been decomposed into multiple graphs from different views according to each specific edge attribute to get the consistent edge adjacency matrices $\mathbb{A}$. Here the multi-view graphs mean there are multiple types for each edge. Hence there are multiple consistent edge attention weights for each edge in each layer.

In this subsection, we present the multi-view spectral graph convolution layer to aggregate the node information over all first-order neighbors from multiple views. In layer l, the new spectral graph convolution for the constrained graph signal in view i is defined as:

$$X_i^{l+1} = \sigma(\Phi_i^l X^l \Theta_i^l) = \sigma(\tilde{D}^{-\frac{1}{2}} \tilde{A}_i^l \tilde{D}^{-\frac{1}{2}} X^l \Theta_i^l), \tag{3.4}$$

where $\tilde{A}_i^l = A_i^l + r_i I$, $\tilde{D}$ is the corresponding normalized degree matrix of $\tilde{A}_i^l$, the ratio r_i is the weight of the self-loop edges of view i, σ is an activation function. Then the outputs from multiple views are concatenated:

$$X^{l+1} = Concate(X_1^l, X_2^l, ..., X_k^l). \tag{3.5}$$

In each view, the spectral GCN layer aggregates the features of each node with all of its neighbors based on consistent attention weights which represent the "strengths" of node-to-node interactions for one edge attribute. Since we have multi-view spectral graph convolutions, the proposed model explores the multiple types of "strengths"

between two nodes in molecules.

In summary, we describe the computation of our proposed method in Algorithm 1. Here the consistent edge attention matrix Φ_i^l is calculated for each view i. The different edge types in different views will derive particular meaningful node embeddings, which are concatenated to develop a complete node feature matrix. In the study of chemical compounds, such collection gives different perspectives of atomic interaction and strength of influence.

Algorithm 1 Consistent Edge-Aware Multi-View Spectral Graph Convolutional Layer l

0: **Input:** view $i \in [1, k]$; layer l; Binary Tensors B_i; Node Feature Matrix X^l
0: **Trainable Parameters:** Edge Mapping Dictionaries M_i^l; Self-attention Weights r_i^l; Parameters Matrices $\Theta_i^l, i = 1, 2, ..., k$
0: **Multi-View Spectral Graph Convolutions:**
0: **for** $i = 1$ to k **do**
0: $A_i^l = B_i \cdot M_i^l$ – Consistent Edge Mapping
0: $\tilde{A}_i^l = A_i^l + r_i^l I$
0: $\Phi_i^l = (\tilde{D}_i^l)^{-1/2} \tilde{A}_i^l (\tilde{D}_i^l)^{-1/2}$ where $\tilde{D}_i^l$ = diag(row-sum $(\tilde{A}_i^l)$) – Diagonal Degree Matrix
0: $X_i^l = \Phi_i^l X^l \Theta_i^l$
0: **end for**
0: **Output:** $X^{l+1} = Concate(X_1^l, X_2^l, ..., X_k^l)$ =0

Graph Invariance and Varying Graph Size. The consistent edge attention matrices imply the multiple strengths of connection and interaction between nodes. The attention weights are conditioned on edge mapping dictionaries instead of the neighborhood order. These edge mapping dictionaries result in a homogeneous view for local graph neighborhoods. These weights are shared over all molecular graphs and not restricted within one graph, which enables us to extract the local stationarity property of the input data by revealing local features that are shared across all graphs. Hence, our model has been designed to produce invariant features by the consistent

edge attention to solve graph invariance problem [38]. In addition, for each edge attention layer, the number of parameters in each attention matrix is the number of edge attribute weights occurred in each dictionary instead of the number of edges. Thus, the number of parameters in *EAGCN* is small and insensitive to the varying graph size, which almost has no influence on the complexity of the model.

Interaction between Substructures. From equation (3.4), for each node, the information is exchanged only with its neighbors within a graph convolution layer. However, if we consider such information propagation from layer to layer, the attentions from higher layers learn the interactions of substructures. The attention weights in Layer 1 represent the atomic interaction between two atoms, while the attention weights in Layer 2 learn the interaction between two substructures centered at two atoms because the information from neighbors is already gathered from the second order neighbors i.e. neighbors' neighbors. Thus the attention weights are capable of characterizing substructure within different scopes to learn the environment information.

Spectral Analysis

The graph convolutional operation presented in subsection 3.4.2 is motivated via the multiplication of a constrained graph signal $\mathbb{A}$ with a new spectral filter. Here we will explain the new spectral filter on feature matrix X^l and consistent edge adjacency matrix A_i^l in view i and layer l. Since the spectral analysis is consistent for both layers and views, so we omit the layer superscript and view subscript in this subsection.

The Laplacian of Consistent Edge Adjacency Matrix An essential tool in spectral graph analysis is the graph Laplacian matrices [13]. We get the normalized graph Laplacian matrix for consistent edge attention matrix, defined as $L = I - D^{-1/2}AD^{-1/2}$, where I is the identity matrix, D is a diagonal degree matrix which contains information about the degree of each vertex. Firstly, we prove the graph Laplacian matrix of consistent edge adjacency matrix satisfies the following properties:

1. For every vector $x \in \mathbb{R}^n$, we have $x^T Lx \geq 0$, which means L is a positive semi-definite matrix.

2. L is a symmetric matrix with n non-negative, real-valued eigenvalues $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n = \lambda_{max} \leq 2$.

Proof: [15] mentions that L is a real symmetric positive semi-definite matrix. However, there isn't theoretical proof for this claim. We provide the detailed proof here.

Let L_e be the Laplacian of graph G on n vertices consisting of just the edge attribute e. By additivity, the graph Laplacian $L = \sum_{e \in E} L_e$, each L_e may have different values by varying weight defined in our dictionary. Without loss of generality, we study two vertices v_1, v_2 in the weighted graph, let e be the edge of (v_1, v_2). By defining the corresponding adjacency matrix as $\begin{bmatrix} w_{11} & w_{12} \\ w_{12} & w_{22} \end{bmatrix}$, degree matrix $D =$

$\begin{bmatrix} w_{11}+w_{12} & 0 \\ 0 & w_{12}+w_{22} \end{bmatrix}$, we have:

$$D^{-1/2}AD^{-1/2} = \begin{bmatrix} \frac{w_{11}}{w_{11}+w_{12}} & \frac{w_{12}}{\sqrt{w_{12}+w_{22}}\sqrt{w_{11}+w_{12}}} \\ \frac{w_{12}}{\sqrt{w_{12}+w_{22}}\sqrt{w_{11}+w_{12}}} & \frac{w_{22}}{w_{12}+w_{22}} \end{bmatrix}.$$

The Laplacian matrix on edge e will be:

$$L_e = I - D^{-1/2}AD^{-1/2} = \begin{bmatrix} \frac{w_{11}}{w_{11}+w_{12}} & \frac{w_{12}}{\sqrt{w_{12}+w_{22}}\sqrt{w_{11}+w_{12}}} \\ \frac{w_{12}}{\sqrt{w_{12}+w_{22}}\sqrt{w_{11}+w_{12}}} & \frac{w_{22}}{w_{12}+w_{22}} \end{bmatrix}.$$

Then for any vector $x \in \mathbb{R}^n$, we get:

$$x^T L_e x = (\frac{\sqrt{w_{12}}}{\sqrt{w_{11}+w_{12}}}x_1 - \frac{\sqrt{w_{12}}}{\sqrt{w_{12}+w_{22}}}x_2)^2 \geq 0.$$

If $x^T L_e x \geq 0$ holds for all non-zero x in R^n, the matrix is said to be a positive semi-definite matrix. It follows immediately that the Laplacian of the whole graph is positive semi-definite. $x^T L x = x^T(\sum_{e\in E} L_e)x = \sum_{e\in E} x^T L_e x \geq 0$, which implies that the symmetric real matrix L is a positive semi-definite matrix [13]. In addition, we also know the eigenvalues of L satisfy $0 = \lambda_1 \leq \lambda_2 \leq \cdots \leq \lambda_n = \lambda_{max} \leq 2$ [13].

Spectral Convolutions on Graph Signals Based on spectral theorem, a positive semi-definite matrix can always be diagonalized using a basis of eigenvectors, so it can be written $L = U\Lambda U^T$ for a unitary matrix U and a diagonal matrix $\Lambda = diag(\lambda_i)$.

Notice that we have $(U\Lambda U^T)^s = U\Lambda^s U^T$, which is the basis of S-localized convolution in graph convolutions analysis.

In spectral convolutions on graphs, the convolution operator on graph in the Fourier domain is defined as $x * y = U((U^T x) \odot (U^T y))$, where $\odot$ is the element-wise Hadamard product. The graph Fourier transform of a graph signal x is defined as $\hat{x} = U^T x \in \mathbb{R}^n$. Here the $U^T y$ and $\odot$ operations are the filtering of the $U^T x$ in the Fourier domain. Thus we can define a signal x with a filter g_θ in the Fourier domain as:

$$y = g_\theta(L)x = g_\theta(U\Lambda U^T)x = U g_\theta(\Lambda)U^T x, \tag{3.6}$$

here θ is a vector of Fourier coefficients. However, the cost of the filtering operation on a signal x is high with $O(n^2)$ because of the multiplication with the Fourier basis U. For solving this problem, some approaches [15, 80] are proposed by parametrizing $g_\theta(L)$ as a polynomial function, such as the Chebyshev polynomials.

The Chebyshev polynomials are defined as $T_s(x) = 2xT_{s-1}(x) - T_{s-2}(x)$, where $T_0(x) = 1$ and $T_1(x) = x$. Hence the $g_\theta(L)$ can be computed recursively from L. We can derive a well approximated S-th order polynomial of a convolution of a signal x with a filter g_θ:

$$g_\theta(\Lambda) \approx \sum_{s=0}^{S-1} \theta_s T_s(\tilde{\Lambda}), \tag{3.7}$$

here we use re-scaled matrix $\tilde{\Lambda} = \frac{2}{\lambda_{max}}\Lambda - I$ to help to guarantee that the eigenvalues of polynomial approximation are rightly in the range $[-1, 1]$, where the $\lambda_{max} \approx 2$. Here we have the $\tilde{L} = L - I = -D^{-1/2}AD^{-1/2}$.

Imagine that we limited the layer-wise convolution operation to $S = 1$, Chebyshev

approximation is linear w.r.t. the graph Laplacian spectrum, specifically:

$$\begin{aligned} y &= g_\theta(L)x = U g_\theta(\Lambda) U^T x \\ &\approx U(\theta_0 + \theta_1 \tilde{\Lambda}) U^T x = \theta_0 x + \theta_1 \tilde{L} x \\ &= \theta_0 \cdot I \cdot x + \theta_1 (-D^{-1/2} A D^{-1/2}) x. \end{aligned}$$

The spectral filter in traditional GCN [44] is derived from the first order Chebyshev approximation which also relies on these two coefficients - the 0^{th} order coefficient θ_0 and 1^{st} order coefficient θ_1. However, these two coefficients are enforced to be $\theta_0 = -\theta_1$, which is a rough approximation.

Spectral Filtering with Fine Approximation In the proposed approach, we relax the "$\theta_0 = -\theta_1$" requirement and propose a new spectral filter with a fine approximation for the constrained graph signals, resulting in more general and effective parameterization of the GCN network.

Without specifying the negative-related parameters, we want to have a more general choice of parameters by using a parameter θ and a real-valued ratio r to satisfy the following conditions:

$$\begin{cases} \theta_0 = r\theta \\ \theta_1 = -\theta, \end{cases}$$

here ratio r is the weight of the self-loop edges which will be proved in the coming

paragraph. Then the expression will come out as:

$$\begin{aligned} y &\approx r\theta \cdot I \cdot x + \theta(D^{-1/2}AD^{-1/2})x \\ &= \theta(rI + D^{-1/2}AD^{-1/2})x. \end{aligned}$$

Different from the traditional spectral analysis of $I + D^{-1/2}AD^{-1/2}$, the weighted graph we learned each time will be added a weighted self-connection. Hence, the eigenvalues will be nested in the range $[r-1, r+1]$. Note that when the ratio becomes 1, it comes to be the case of the traditional one. In order to leave the eigenvalues in the range $[0, 1]$ to ensure a more stable deep neural network, we do re-normalization:

$$y \approx \theta(\tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}})x, \tag{3.8}$$

where $\tilde{A} = A + rI$, $\tilde{D}$ is the corresponding normalized degree matrix of $\tilde{A}$. One interesting finding is that the ratio r can be considered as the weight of the self-loop edges as illustrated in Algorithm 1. By adding this parameter into neural network, each node has the ability to control the importance of original information and features in itself. In summary, this propagation rule for consistent edge attention matrix is derived based on a first-order approximation with more plausible assumptions and reasonable parameterization compared to the work in [44].

Then we get a generalized formula to a signal $X \in \mathbb{R}^{n \times f}$ with f input channels and filter parameters matrix $\Theta \in \mathbb{R}^{f \times f'}$ with f' filters:

$$Y = \tilde{D}^{-\frac{1}{2}}\tilde{A}\tilde{D}^{-\frac{1}{2}}X\Theta = \Phi X\Theta, \tag{3.9}$$

thus we derive the convolved signal matrix $Y \in \mathbb{R}^{n \times f'}$. If input graph signal X is

the X^l, the output Y of proposed method will be X^{l+1}. In addition, when we extend our analysis to multi-view graph convolutions, a set of parameters $\{r_1, ..., r_k\}$ will be used in our model. Here the new filter with more parameters can benefit the property prediction.

3.4.3 Graph Representation Layer

From the previous sections, the node representations have been generated. Another step is how to get the graph representation using these node representations. Here two ways are provided to get the graph representation. The comparison between them will be given in the experiments section.

Simple Sum

The simple way to get the graph representation is the cumulative sum of all node embeddings or representation as the graph representation:

$$X^{l+1} = Sum(x_1^l, x_2^l, ..., x_n^l) \in \mathbb{R}^{1 \times f_l},$$

where x_i^l means the representation of node i in layer l, n is the number of node and f_l is the dimension of node representation in layer l.

Differentiable Pooling

The differentiable pooling (Diffpool) [95] is proposed to learn the hierarchical representation of graphs. Diffpool designs the graph pooling neural network to generate

the assignment matrix:

$$S^l = softmax(GCN_1^l(A^l, X^l)) \in R^{n_l \times n_{l+1}},$$

where the softmax function is applied in a row-wise fashion, the inputs are the feature matrix X^l and cluster adjacency matrix A^l and GCN_1^l is the traditional graph convolutional network [44] in layer l. Notice that $n_l = n$ in the first layer of Diffpool. In our experiments, since we have multiple consistent attention matrices, the input of first differential pooling layer is $A^l = \sum_{i=1}^{k} \tilde{A}_{att,i}^{l-1}$. Then the new cluster adjacency matrix and the feature (representation) matrix of cluster nodes can be obtained using another GCN_2^l as follows:

$$X^{l+1} = (S^l)^T GCN_2^l(A^l, X^l),$$

$$A^{l+1} = (S^l)^T A^l S^l.$$

This final output embedding is the graph representation $X \in \mathbb{R}^{1 \times f_G}$, here f_G is the dimension of a graph representation.

3.5 Experiments

3.5.1 Benchmark Datasets

Five benchmark datasets [89, 1] (Tox21, Freesolv, Lipophilicity, eSOL and CPAL) are utilized in this study to evaluate the predictive performance of the *EAGCN* model.

Tox21. The original Tox21 data comes from the Toxicology in the 21st century

research initiative. It contains 7831 environmental compounds and drugs as well as their biological outcomes of 12 pathway assays.

Freesolv. Freesolv is a database of experimental and calculated hydration free energies for small neutral molecules in water, along with molecular structures, input files, references, and annotations [62]. It includes a set of 642 neutral molecules which are mostly fragment-like.

Lipophilicity (Lipo). Lipophilicity, curated from ChEMBL database, provides experimental results of octanol/water distribution coefficient of 4200 compounds.

eSOL. eSOL is a database on the solubility of entire ensemble *E.coli* proteins [68] individually synthesized by PURE system that is chaperone free.

CPAL. The "Cytotoxic Profiling of Annotated Libraries Using Quantitative High-Throughput Screening" (CPAL) dataset is provided by Pubchem. The CPAL data can be download from Pubchem website[2].

Four of them except CPAL are downloaded from the MoleculeNet website[3] that hold various benchmark datasets for molecular machine learning. In our modeling processes, we only keep molecules with common atom types, e.g. boron, carbon, nitrogen, oxygen, halogen, phosphorus, sulfur.

3.5.2 Experimental Setup

Our experiments evaluate the property prediction on standard supervised classification and regression tasks. We design our experiments with the goal of verifying the improvement of our method compared with baseline methods, such as GCN [44], and do more in-depth analysis for the graph and atom embeddings.

[2]https://pubchem.ncbi.nlm.nih.gov/bioassay/1296008
[3]http://moleculenet.ai/

The node features and edge attributes are extracted using the RDKit[4], an open source cheminformatics package. RDKit also converts the SMILES string format into RDKit "mol" format, which contains the molecular structure information used to build a molecular graph. Here we ignore the SMILES samples whose structure graphs haven't edges. The edge attributes [14] are shown in Table 3.1. When we build the edge mapping dictionaries for atom pair types, we set a threshold on the frequency of atom pair types for each dataset. For the atom pair types whose frequencies are lower than the threshold, we will set one attention weight for them in the dictionary. Each data is randomly split into three sets: training (80%), validation (10%), and testing (10%). For each experiment, 5 independent runs with different random seeds are performed. The number of multi-view spectral graph convolutional layer is 4. For a fair comparison, the number of layers for GCN baseline is also 4. In addition, when we use the two Diffpool layers. Note that all results are the average of 5 runs and we provide the standard deviations. We use the adaptive moment (Adam) algorithm [41] for training the model and set the learning rate to 0.0001 for freesolv and tox21 datasets and 0.001 for other datasets.

3.5.3 Molecular Property Prediction

We compare our model with several baseline methods which are shown in MoleculeNet [89]. Firstly, the Kernel-SVM and Random Forests (RF) which are the most famous machine learning methods are used here. Secondly, the graph convolutions network (GCN) [44] is the baseline for the comparison. In addition, we also compare with other models which leverage the edge attributes. Weave model [38] is similar to

[4]http://www.rdkit.org/

graph convolutions. The weave featurization encodes both local chemical environment and connectivity of atoms in a molecule. Message passing neural network (MPNN) [23] is a generalized model of GCN, which learns a message passing algorithm and aggregation procedure to compute a function of their entire input graph. Graph Attention Networks (GAT) is the benchmark of the attention-based GCN, which determines the weight for an edge based on the similarity of the two nodes.

Regression Analysis

Solubility and lipophilicity are basic physical chemistry properties that are important for understanding how molecules interact with solvents. Table 3.2 reports RMSE results of five different baseline models and our EAGCN model. Firstly, the comparison between EAGCN$_{sum}$ and EAGCN$_{diffpool}$ shows the simple sum operation used in graph representation layer has slightly better performance than the complex Diffpool operation. The reason is Diffpool method [95] was used in the larger graphs, which is more suitable to learn the hierarchical graph representation. However, when the inputs are the small molecular graphs with different sizes and shapes, the simple sum operation is more suitable to get graph representation than Diffpool.

Second, we compare with the best baseline models from the survey [89] with our proposed EAGCN$_{sum}$ for the regression task. In Lipophilicity dataset, EAGCN$_{sum}$ achieves about 6.7% and 6.3% performance increases comparing GCN which is the best baseline on both the validation and test datasets shown in [89]. For the Freesolv dataset, we improve upon GCN by a margin of 13.7% and 13.8% on the validation and test datasets. For the eSOL dataset, EAGCN$_{sum}$ improves upon MPNN by a margin of 37.2% on the validation dataset and by a margin of 9.0% on test dataset.

Table 3.2: Regression results for the three benchmark datasets. EAGCN[1] is the EAGCN$_{sum}$ and EAGCN[2] is the EAGCN$_{diffpool}$.

Method	Regression (RMSE)					
	Lipo		Freesolv		eSOL	
	Valid	Test	Valid	Test	Valid	Test
RF	0.85 ± 0.02	0.85 ± 0.03	2.31 ± 0.82	1.62 ± 0.14	1.27 ± 0.07	1.18 ± 0.11
Weave	0.85 ± 0.10	0.88 ± 0.09	1.63 ± 0.25	1.37 ± 0.19	0.90 ± 0.08	0.85 ± 0.03
MPNN	0.81 ± 0.05	0.80 ± 0.05	1.26 ± 0.42	1.09 ± 0.19	0.86 ± 0.00	0.67 ± 0.06
GAT	0.85 ± 0.02	0.84 ± 0.01	1.73 ± 0.25	1.45 ± 0.25	0.86 ± 0.00	1.00 ± 0.20
GCN	0.60 ± 0.03	0.64 ± 0.03	1.24 ± 0.26	0.94 ± 0.10	0.86 ± 0.00	0.80 ± 0.17
EAGCN[1]	$\mathbf{0.56 \pm 0.03}$	$\mathbf{0.60 \pm 0.03}$	$\mathbf{1.07 \pm 0.28}$	$\mathbf{0.81 \pm 0.04}$	$\mathbf{0.54 \pm 0.06}$	$\mathbf{0.61 \pm 0.10}$
EAGCN[2]	0.61 ± 0.08	0.63 ± 0.04	1.14 ± 0.27	0.86 ± 0.09	0.59 ± 0.06	0.65 ± 0.06

Since the EAGCN model achieves the best performance, using the bond lengths and bond energies information does benefit the molecular property prediction task.

EAGCN shows strong performance on all datasets. Given the size of FreeSolv dataset is only around 600 compounds, our model still reaches excellent performances by training on limited samples. From the table, graph-based methods, such as *EAGCN* and graph convolutional model, exhibit significant boosts over tasks, indicating the advantages of the learnable featurization. In these three datasets, data-driven methods outperform physical algorithms with moderate amounts of data. These results suggest that graph convolutions based approaches will become increasingly important for the property prediction.

Classification Analysis

Table 3.3 reports ROC-AUC results of six different baselines on Tox21 and CPAL datasets. For Tox21 dataset, the GAT model achieves the best performances comparing all other baselines on the test and validation datasets. Our EAGCN$_{sum}$ model improves upon GAT by a margin of 1.2% and 2.4% on the validation and test datasets. For CPAL dataset, EAGCN$_{sum}$ model improves upon GCN by a margin of 2.3% on validation dataset and achieves about 2.4% comparing GAT on the test dataset. We conclude that EAGCN$_{sum}$ always achieves reasonable and excellent performance for prediction of properties. After checking the last two rows in Table 3.3, EAGCN$_{sum}$ performance is still slightly better than EAGCN$_{diffpool}$, which is consistent with the regression task. There are many applications which can benefit from our model. For example, the classification model building upon the Tox21 can be utilized to identify new compounds with potential liability and prioritize specific compounds for more

Table 3.3: Classification results for the two benchmark datasets. EAGCN[1] is the EAGCN$_{sum}$ and EAGCN[2] is the EAGCN$_{diffpool}$.

Method	Classification (ROC-AUC)			
	Tox21		CPAL	
	Valid	Test	Valid	Test
Kernel-SVM	0.77 ± 0.01	0.77 ± 0.02	0.79 ± 0.02	0.76 ± 0.06
RF	0.76 ± 0.01	0.77 ± 0.01	0.77 ± 0.05	0.74 ± 0.06
Weave	0.81 ± 0.01	0.83 ± 0.01	0.78 ± 0.06	0.77 ± 0.08
MPNN	0.80 ± 0.02	0.82 ± 0.02	0.83 ± 0.04	0.78 ± 0.07
GAT	0.84 ± 0.02	0.84 ± 0.02	0.81 ± 0.04	0.83 ± 0.05
GCN	0.83 ± 0.01	0.84 ± 0.01	0.88 ± 0.02	0.80 ± 0.07
EAGCN[1]	$\mathbf{0.85 \pm 0.01}$	$\mathbf{0.86 \pm 0.01}$	$\mathbf{0.90 \pm 0.02}$	$\mathbf{0.85 \pm 0.06}$
EAGCN[2]	0.85 ± 0.01	0.85 ± 0.01	0.87 ± 0.02	0.83 ± 0.07

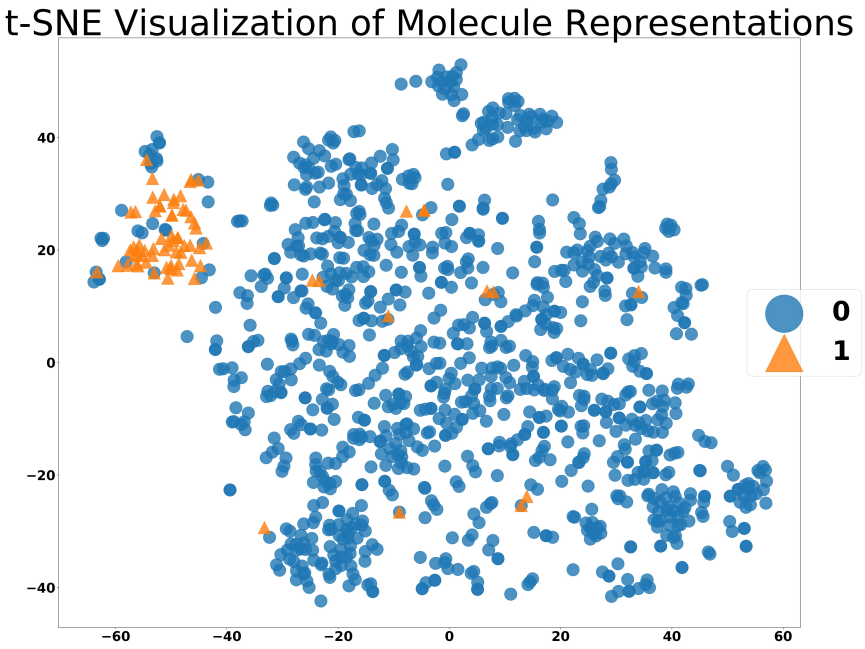
Table 3.4: The paired t-test for model comparison. Here $EAGCN_1$ is $EAGCN_{sum}$ and $EAGCN_2$ is $EAGCN_{diffpool}$.

Models		p-value	
		EAGCN$_1$ vs GCN	EAGCN$_1$ vs EAGCN$_2$
Lipo	Validation	0.0373	0.3105
	Testing	0.0232	0.1992
Freesolv	Validation	0.0297	0.1966
	Testing	0.0247	0.1010
eSOL	Validation	0.0265	0.0562
	Testing	0.0106	0.3697
Tox21	Validation	0.0002	0.0921
	Testing	0.0156	0.0719
CPAL	Validation	0.0129	0.0923
	Testing	0.0305	0.2862

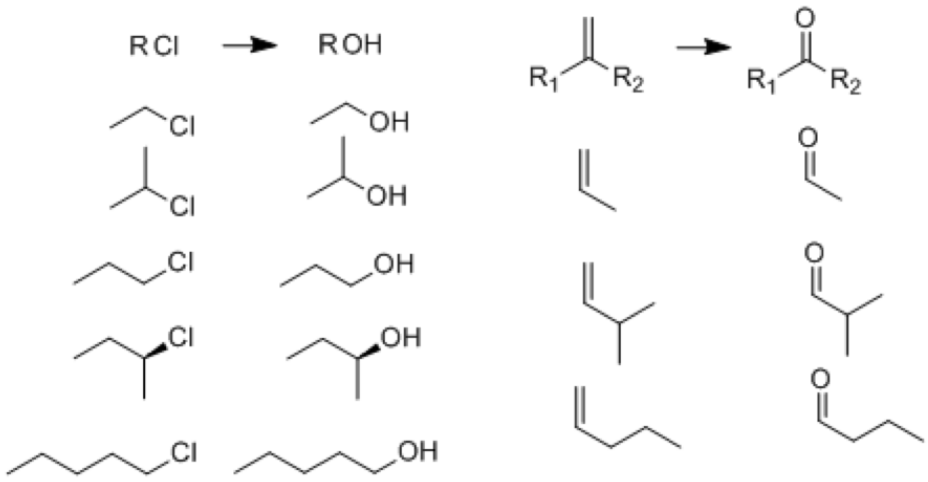
extensive toxicological evaluation.

The t-test Results

In Table 3.4, the mean and standard deviation of model performance are computed according to the five different settings using different data splits. To compare if the two models are significantly different, we do a paired t-test on both validation set and test set. The null hypothesis H_0 is the two models are the same. In Table 3.5, the p-values in the column of "EAGCN$_{sum}$ vs GCN" show that we should reject H_0 because all values are less than 0.05. Hence the EAGCN$_{sum}$ model is significantly different from GCN model at 0.05 level. In the column of "EAGCN$_{sum}$ vs EAGCN$_{diffpool}$", the p-values indicate that two different graph representation methods have no significant difference at 0.05 level.



(a) Visualization of molecular graph embeddings in CPAL.



(b) Matched molecular pair (MMP) examples.

Figure 3.4: Graph representation visualization and matched molecular pair examples.

3.5.4 Molecular Graph Embedding Analysis

In Figure 3.4(a), the representations of molecular graphs utilize t-SNE [56] to reduce the dimensionality of representations and visualize atom embeddings and potential relations in the projected 2-dimensional space. The striking feature is that most points of graph representations can be clearly separated. The points labeled by "1" (triangle) are restricted to the top-left corner of the projection plane. Note that the points with label "0" are the sampling points from the original dataset.

Since many molecular graphs have similar structures but different property values, it is a challenge to predict their property accurately. For example, only one atom of the first three molecules in Table 3.5 is different. In addition, only one bond between the two molecules of the second block is different. Different bond orders and atoms do lead to different property values. After checking the true properties and prediction values, we conclude that our predictions for these pairs are pretty accurate even they have similar structures but different properties.

The potential relationships cross molecules are very similar with the transformation between the word vectors in Natural Language Processing (NLP). The word embeddings often preserve the relations in the projected 2-dimensional space. For example, the relationship between male and female [61] is automatically learned. The "Queen" embedding can be derived from "King - Man + Woman". In the field of chemoinformatics, chemists are often interested in comparing properties of two molecules that differ only by a single chemical transformation, such as the substitution of a hydrogen atom by a chlorine one. Such pairs of compounds are usually referred as matched molecular pairs (MMP), as demonstrated by Figure 3.4 (b). We are interested in checking whether our atomic vector representations can capture the sin-

Table 3.5: The molecules with similar structure but different property in the Freesolv dataset. "=" is double bond and "#" is triple bond.

Name	SMILES	Label	Prediction
1-chloro-2-methyl-benzene	**Cc1ccccc1Cl**	-1.14	-1.14
2-methylaniline	**Cc1ccccc1N**	-5.53	-5.53
o-cresol	**Cc1ccccc1O**	-5.9	-5.69
pent-1-yne	**CCCC#C**	0.01	0.10
pent-1-ene	**CCCC=C**	1.68	1.49
pent-1-ene	CCCC=C	1.68	1.49
butanal	CCCC=O	-3.18	-3.21
2-chloro-2-methyl-propane	CC(C)(C)Cl	1.09	1.43
2-methylpropan-2-ol	CC(C)(C)O	-4.47	-4.35
isobutane	CC(C)C	2.3	2.17
2-bromopropane	CC(C)Br	-0.48	-0.40
1-ethyl-2-methylbenzene	CCc1ccccc1C	-0.85	-0.91
2-ethylphenol	CCc1ccccc1O	-5.66	-5.62
pent-1-yne	CCCC#C	0.01	0.10
butanenitrile	CCCC#N	-3.64	-3.78
hex-1-ene	CCCCC=C	1.58	1.70
pentanal	CCCCC=O	-3.03	-2.95

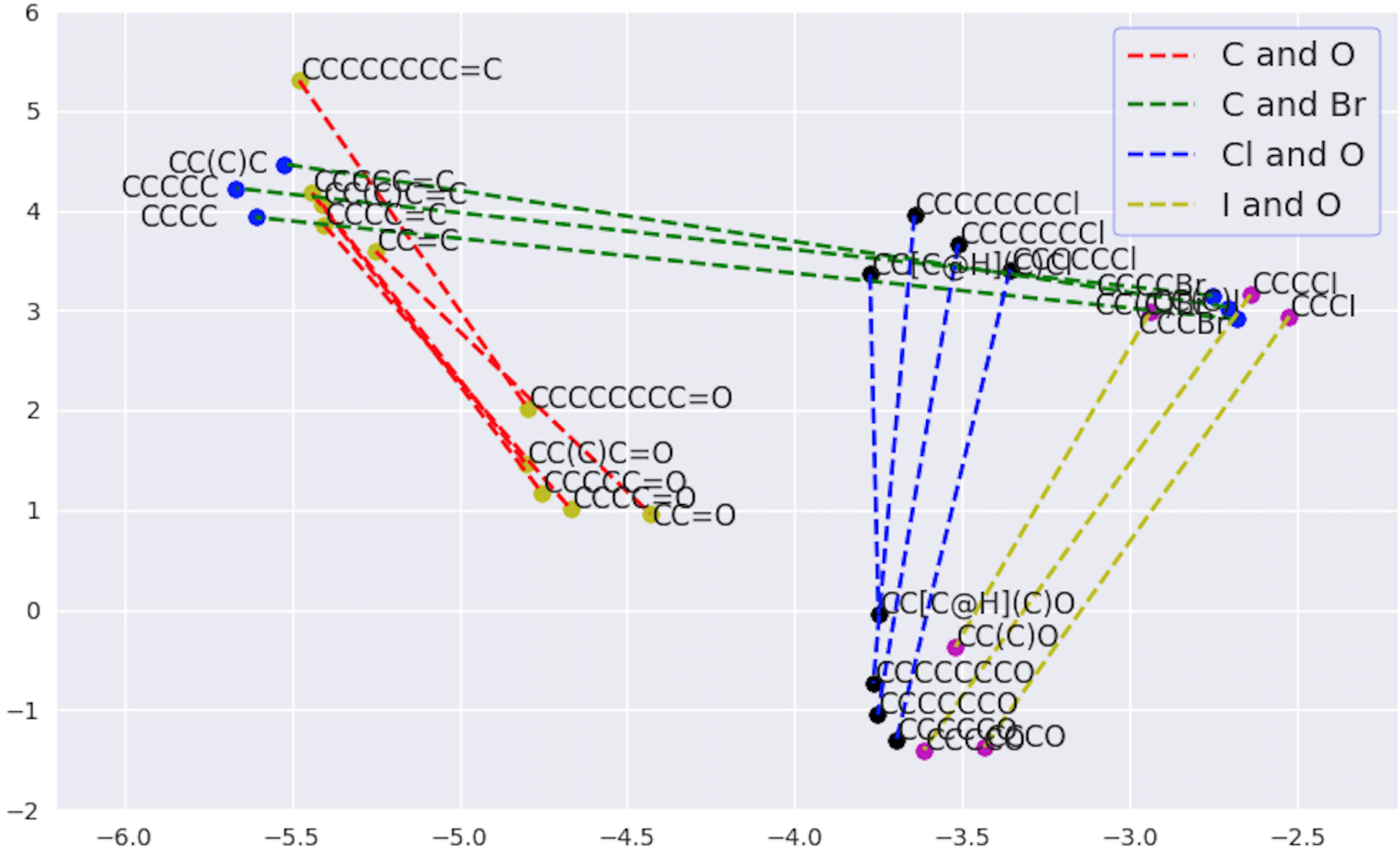


Figure 3.5: Visualization of molecular representation using PCA in Freesolv dataset.

gularity relationship among certain matched molecule pair in the investigated dataset and further provide visual interpretation. Therefore, we plot the two-dimensional PCA plots for several subsets of matched molecule pairs in Figure 3.5. Similar to the vector analyses in Word2Vec, the vector transformations between each matched pair are mapped to 2D PCA plot.

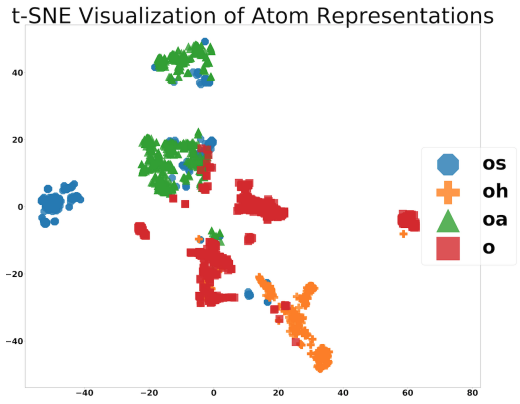
As illustrated in Figure 3.5, we obtain a series of relatively parallel vector transformation line within each subset of matched molecule pair, which may imply that our EAGCN model captures the implicit relationships of each molecular pair and such relationships can be further applied to predict the relevant chemical properties of related analogs. For instance, when chloro substitutions are replaced with the hydroxy group in the first MMP example, the resulting transforming vectors are almost parallel and all point toward the same direction, which implies these similar structural changes lead to similar changing effects on solubility property.

3.5.5 Atom Subtype Analysis

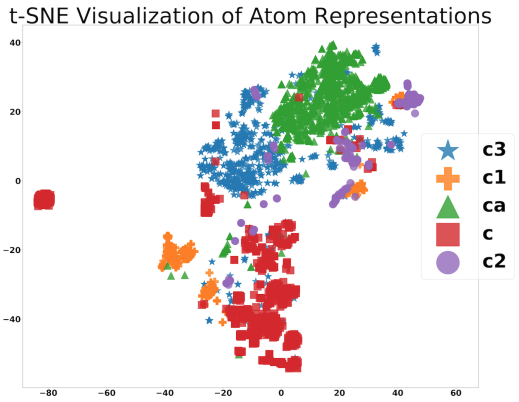
In our study, atom types of molecules are determined prior to the graph convolutional network modeling. Atom types are classified in Table 3.6 based on their elements, atomic connectivity and hybridization states. The atomic representation vectors are collected from the output of the last multi-view spectral graph convolutional layer in our model. We visualize these high-level atom representations learned by the graph convolutional neural nets by t-SNE [56]. t-SNE is a nonlinear dimensionality reduction technique to embed the high-dimensional atom vector representation to a two-dimensional space. It constructs the probability distribution over data points in both original high-dimensional space and the reduced low-dimensional space, and

Table 3.6: Atom Subtype Definition.

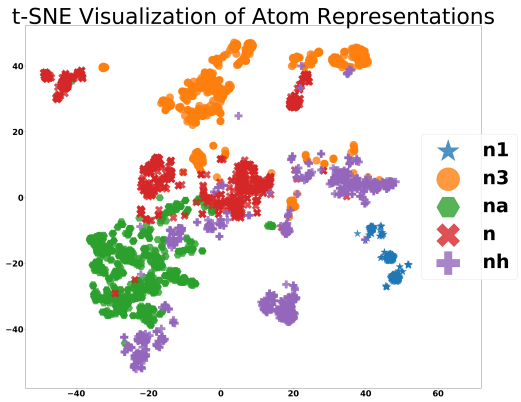
Atom	Subtype	Definition
O	O	carbonyl oxygen
	Oa	sp2-hybridized oxygen in the aromatic ring
	Oh	sp3-hybridized oxygen in alcohol
	Os	sp3-hybridized oxygen in ether or ester
C	C	carboxylate, carbonyl and thion carbon
	C1	sp-hybridized carbon
	C2	sp2-hybridized noaromatic carbon with one substitution
	C3	sp3-hybridized carbon
	Ca	aromatic carbon
N	N	sp2-hybridized with 3 substituents and sp2-hybridized nitrogen in base NH2 group
	N1	sp1-hybridized nitrogen
	N3	sp3-hybridized nitrogen
	Na	sp2-hybridized aromatic nitrogen
	Nh	sp2-hybridized nitrogen in amide group



(a) Visualization of O subtypes.



(b) Visualization of C subtypes.



(c) Visualization of N subtypes.

Figure 3.6: Visualizations of atom embeddings using t-SNE in Lipo. Points are colored by atom subtypes defined by Chemist.

then minimizes the Kullback-Leibler divergence between the two distributions.

As shown in Figure 3.6, vector representations of various oxygen and carbon atom subtypes from EAGCN model in Lipo dataset are mapped into 2D t-SNE visualization plots. In each t-SNE plot, projected atom type representations are colored by atom subtypes specified in Table 3.6. For instance, in Figure 3.6 (a), depending on their chemical bonding environment, oxygen atoms within molecules of the Lipo dataset can be classified as the following four subtypes: sp3-hybridized oxygen in alcohol (oh), sp2-hybridized oxygen in the aromatic ring (oa), carbonyl oxygen (o), sp3-hybridized oxygen in ether or ester (os). Interestingly, those different atom subtype representations learned from the graph convolutional network model are relatively well clustered within the t-SNE plot, which is consistent with the predefined atom subtype based on the chemical intuition. We observe the similar effects for the carbon and nitrogen atom subtypes, as demonstrated by Figure 3.6 (b) and (c). Hence the embeddings of atoms from EAGCN model do learn the atom type and subtype information.

3.5.6 Interpretation of Attention Weights

In physical chemistry, the edge mapping dictionaries learn multiple strengths of influence from neighbors, which gives some insights on atomic interaction. In Figure 3.7, the edge mapping dictionaries are visualized by heatmaps. The darkness of a block corresponds to the attention weight value. The attention weights in Layer 1 represent the atomic interaction between two atoms, while the attention weights in Layer 2 learn the interaction between two substructures centered at two atoms. Hence the attention weights in different layers have different values. In Figure 3.7, both $C - O$

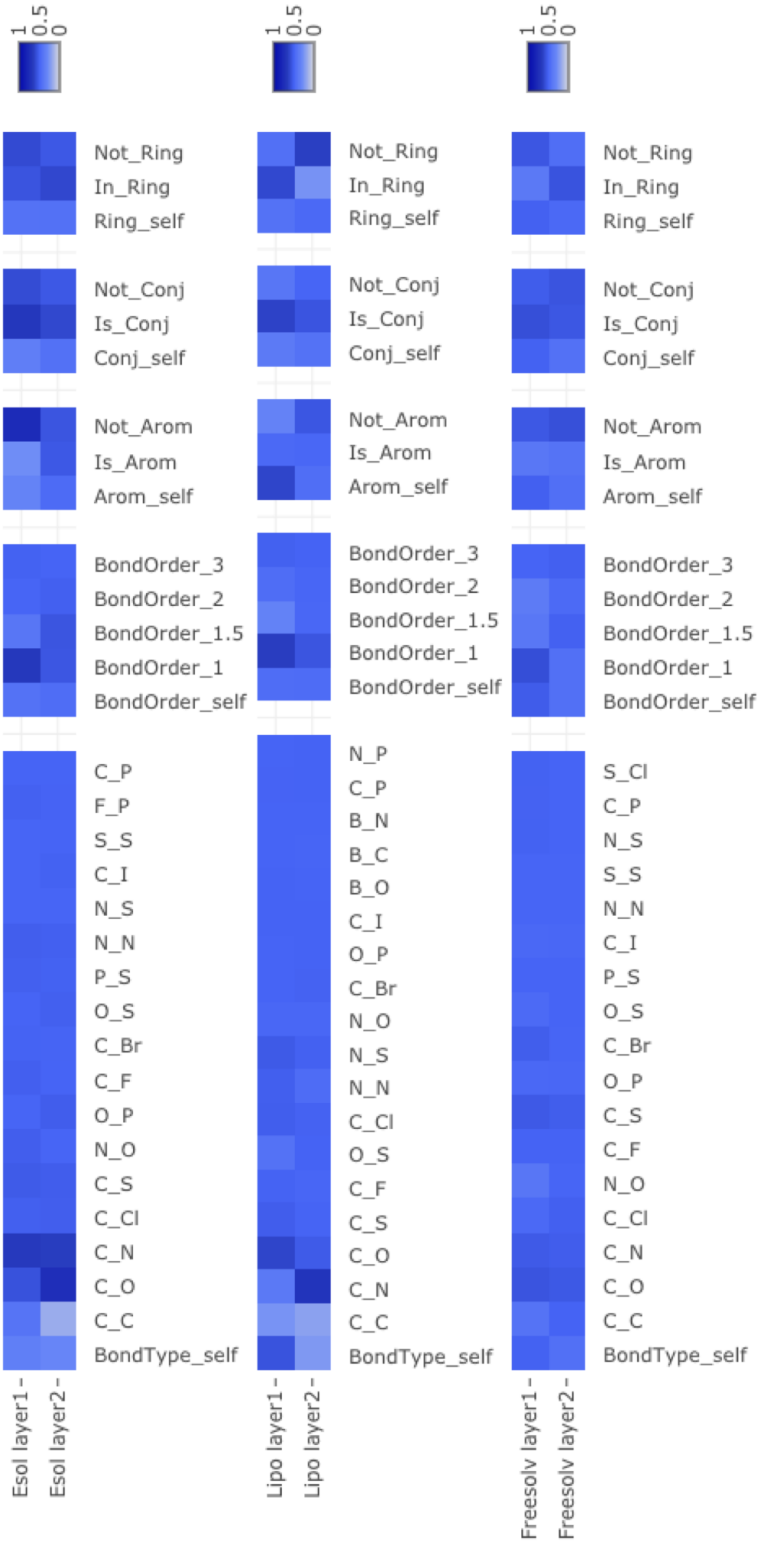


Figure 3.7: Visualization of the learned edge attention weights. The labels are the edge types. The self-loop weights r_i^l are labeled with "_self".

and $C-N$ attention weights in first layer for eSOL and Freesolv have higher attention values. This is consistent with the analysis in Table 3.5, which shows O and N are highly related with the solubility property. For edge mapping dictionaries of Lipo, the $C-N$ has a large weight in second layer. That means the substructure around C has a significant strength of influence to the substructure around N for the property.

Chapter 4

Discrete Graph Structure Learning to Simultaneously Forecast Multiple Time Series

4.1 Motivation

Time series data are widely studied in science and engineering that involve temporal measurements. Time series forecasting is concerned with the prediction of future values based on observed ones in the past. It has played important roles in climate studies, market analysis, traffic control, and energy grid management [59] and has inspired the development of various predictive models that capture the temporal dynamics of the underlying system. These models range from early autoregressive approaches [29, 2] to the recent deep learning methods [74, 51, 96, 101].

The forecasting and analysis of univariate time series (a single longitudinal variable) has been extended to multivariate time series and multiple (univariate or mul-

tivariate) time series. Multivariate forecasting models find strong predictive power in stressing the interdependency (and even causal relationship) among the variables. The vector autoregressive model [29] is an example of multivariate analysis, wherein the coefficient magnitudes offer hints into the Granger causality [27] of one variable to another.

For multiple time series, pairwise similarities or connections among them are also explored to improve the forecasting accuracy [96]. An example is the traffic network where each node denotes a time series recording captured by a particular sensor. The spatial connections of the roads offer insights into how traffic dynamics propagates along with the network. Several graph neural network (GNN) approaches [74, 51, 96, 101] have been proposed recently to leverage the graph structure for forecasting all time series.

The graph structure however is not always available or it may be incomplete. There could be several reasons, including the difficulty in obtaining such information or a deliberate shielding for the protection of sensitive information. For example, a dataset comprising sensory readings of the nation-wide energy grid is granted access to specific users without disclosure of the grid structure. Such practical situations incentivize the automatic learning of the hidden graph structure jointly with the forecasting model.

Because GNN approaches show promise in forecasting multiple interrelated time series, in this approach, we are concerned with structure learning methods applied to the downstream use of GNNs. A prominent example is a recent work LDS [21], which is a meta-learning approach that treats the graph as a hyperparameter in a bilevel optimization framework [20]. Specifically, let X_{train} and X_{val} denote the training and the validation sets of time series respectively, $A \in \{0, 1\}^{n \times n}$ denote the

graph adjacency matrix of the n time series, w denote the parameters used in the GNN, and L and F denote the loss functions used during training and the validation respectively (which may not be identical). LDS formulates the problem as learning the probability matrix $\theta \in [0,1]^{n \times n}$, which parameterizes the element-wise Bernoulli distribution from which the adjacency matrix A is sampled:

$$\begin{aligned} \min_{\theta} \quad & {}_{A \sim (\theta)}[F(A, w(\theta), X_{\text{val}})], \\ \text{s.t.} \quad & w(\theta) = \arg\min_{w} \ {}_{A \sim (\theta)}[L(A, w, X_{\text{train}})]. \end{aligned} \tag{4.1}$$

Formulation (4.1) gives a bilevel optimization problem. The constraint (which by itself is an optimization problem) defines the GNN weights as a function of the given graph, so that the objective is to optimize over such a graph only. Note that for differentiability, one does not directly operate on the discrete graph adjacency matrix A, but on the continuous probabilities θ instead.

LDS has two drawbacks. First, computation is expensive. The derivative of w with respect to θ is computed by applying the chain rule on a recursive-dynamics surrogate of the inner optimization argmin. Applying the chain rule on this surrogate is equivalent to differentiating an RNN, which is either memory intensive if done in the reverse mode or time consuming if done in the forward mode, when unrolling a deep dynamics. Second, it is challenging to scale. The matrix θ has $\Theta(n^2)$ entries to optimize and thus the method is hard to scale to increasingly more time series.

In light of the challenges of LDS, we instead advocate an unilevel optimization:

$$\min_{w} \quad {}_{A \sim (\theta(w))}[F(A, w, X_{\text{train}})]. \tag{4.2}$$

Formulation (4.2) trains the GNN model as usual, except that the probabilities θ (which parameterizes the distribution from which A is sampled), is by itself parameterized. We absorb these parameters, together with the GNN parameters, into the notation w. We still use a validation set X_{val} for usual hyperparameter tuning, but these hyperparameters are not θ as treated by (4.1). In fact, formulation (4.1) may need a second validation set to tune other hyperparameters.

The major distinction of our approach from LDS is the parameterization $\theta(w)$, as opposed to an inner optimization $w(\theta)$. In our approach, a modeler owns the freedom to design the parameterization and better control the number of parameters as n^2 increases. To this end, time series representation learning and link prediction techniques offer ample inspiration for modeling. In contrast, LDS is more agnostic as no modeling is needed. The effort, instead, lies in the nontrivial treatment of the inner optimization (in particular, its differentiation).

As such, our approach is advantageous in two regards. First, computation is less expensive, because the gradient computation of an unilevel optimization is straightforward and efficient and implementations are mature. Second, it is better scaled, because the number of parameters does not grow quadratically with the number of time series.

We coin our approach GTS (short for "graph for time series"), signaling the usefulness of graph structure learning for improving time series forecasting. We develop an architecture (see Section 4.3) to learn a GNN model and infer the pairwise structure θ concurrently. Three contributions are:

1. We propose an approach to learn the pairwise interactions among multiple time series. This approach is simpler than a bilevel optimization adopted by meta-

learning approaches. This approach also incorporates a priori knowledge of the interactions, if exists.

2. We demonstrate that simultaneous forecasting of multiple time series improves over independent forecasting, with the use of properly learned graph structure.

3. We also show that the learned structure outperforms a naive neighborhood graph, when used by a GNN for time series forecasting.

4.2 Related Work

Time series forecasting has been studied for decades by statisticians. It is out of the scope of this dissertation to comprehensively survey the literature, but we will focus more on late developments under the deep learning context. Early textbook methods include (vector) autoregressive models [29], autoregressive integrated moving average (ARIMA) [2], hidden Markov models (HMM) [5], and Kalman filters [98]. Generally speaking, these are linear models that use a window of the past information to predict the next time step, although nonlinear versions with parameterization are subsequently developed.

A notable nonlinear extension was the RNN [88], which later evolved into LSTM [33], BiLSTM [73], and GRU [12], which addressed several limitations of the vanilla RNN, such as the vanishing gradient problem. These architectures are hard to parallelize because of the recurrent nature of the forward and backward computation. More recently, Transformer [84] and BERT [17] were developed to address parallelization, by introducing attention mechanisms that simultaneously digested past (and future) information. Although these models are more heavily used for sequence data under

the context of natural language processing, they are readily applicable for time series as well.

Graph neural networks [99, 103, 91] emerged quickly in deep learning to handle graph-structured data. Typically, graph nodes are represented by feature vectors, but for the case of time series, a number of specialized architectures were recently developed; see, e.g., GCRN [74], DCRNN [51], STGCN [96], and T-GCN [101]. These architectures essentially combine the temporal recurrent processing with graph convolution to augment the representation learning of the individual time series.

Graph structure learning (not necessarily for time series) appears in various contexts and thus methods span a broad spectrum. One field of study is probabilistic graphical models and casual inference, whereby the directed acyclic structure is enforced. Gradient-based approaches in this context include NOTEARS [102], DAG-GNN [97], and GraN-DAG [45]. On the other hand, a general graph may still be useful without resorting to causality. LDS [21] is a meta-learning approach that demonstrates to improve the performance of node classification tasks. NRI [43] adopts a latent-variable approach and learns a latent graph for forecasting system dynamics. We have compared our approach with LDS in the preceding section and will compare with NRI in the following one.

4.3 Method

In this section, we present the proposed GTS method, elaborate the model parameterization, and describe the training technique. We also highlight the distinctions from NRI.

Let us first settle the notations. Denote by X the training data, which is a three dimensional tensor, with the three dimensions being feature, time, and the n series. Superscript refers to the series and subscript refers to time; that is, X^i denotes the i-th series for all features and time and X_t denotes the t-th time step for all features and series. There are in total S time steps for training. The model will use a window of T steps to forecast the next τ steps. For each valid t, denote by $\widehat{X}_{t+T+1:t+T+\tau} = f(A, w, X_{t+1:t+T})$ the model, which forecasts $\widehat{X}_{t+T+1:t+T+\tau}$ from observations $X_{t+1:t+T}$, through exploiting the graph structure A and being parameterized by w. Using ℓ to denote the loss function between the prediction and the ground truth, a typical training objective reads

$$\sum_t \ell(f(A, w, X_{t+1:t+T}),\ X_{t+T+1:t+T+\tau}). \tag{4.3}$$

Three remaining details are the parameterization of A, the model f, and the loss ℓ.

4.3.1 Graph Structure Parameterization

The binary matrix $A \in \{0,1\}^{n\times n}$ by itself is challenging to parameterize, because it requires a differentiable function that outputs discrete values $0/1$. A more natural idea is to let A be a random variable of the matrix Bernoulli distribution parameterized by $\theta \in [0,1]^{n\times n}$, so that A_{ij} is independent for all the (i,j) pairs with $A_{ij} \sim (\theta_{ij})$. Here, θ_{ij} is the success probability of a Bernoulli distribution. Then, the training objective (4.3) needs to be modified to

$$_{A\sim(\theta)}\left[\sum_t \ell(f(A, w, X_{t+1:t+T}),\ X_{t+T+1:t+T+\tau})\right]. \tag{4.4}$$

As hinted in Section 4.1, we further parameterize θ as $\theta(w)$, because otherwise the n^2 degrees of freedom in θ render the optimization hard to scale. Such a parameterization, however, imposes a challenge on differentiability, if the expectation (4.4) is evaluated through sample average: the gradient of (4.4) does not flow through A in a usual Bernoulli sampling. Hence, we apply the Gumbel reparameterization trick proposed by [36, 57]: $A_{ij} = ((\log \theta_{ij} + g_{ij})/s)$, where $g_{ij} \sim (0, 1)$ for all i, j. When the temperature $s \to 0$, $A_{ij} = 1$ with probability θ_{ij} and 0 with probability $1 - \theta_{ij}$. In practice, we anneal s progressively in training such that it tends to zero.

For the parameterization of θ, we use a feature extractor to yield a feature vector for each series and a link predictor that takes in a pair of feature vectors and outputs a link probability. The feature extractor maps a matrix X^i to a vector z^i for each i. Many sequence architectures can be applied; we opt for a simple one. Specifically, we perform convolution along the temporal dimension, vectorize along this dimension, and apply a fully connected layer to reduce the dimension; that is, $z^i = \mathrm{FC}(\mathrm{vec}(\mathrm{Conv}(X^i)))$. Note that the feature extractor is conducted on the entire sequence rather than a window of T time steps. Weights are shared among all series.

The link predictor maps a pair of vectors (z^i, z^j) to a scalar $\theta_{ij} \in [0, 1]$. We concatenate the two vectors and apply two fully connected layers to achieve so; that is, $\theta_{ij} = \mathrm{FC}(\mathrm{FC}(z^i \| z^j))$. The last activation needs to be a sigmoid.

Details so far are summarized in the top part of Figure 4.1. The bottom part is the forecasting model f, whose details come next.

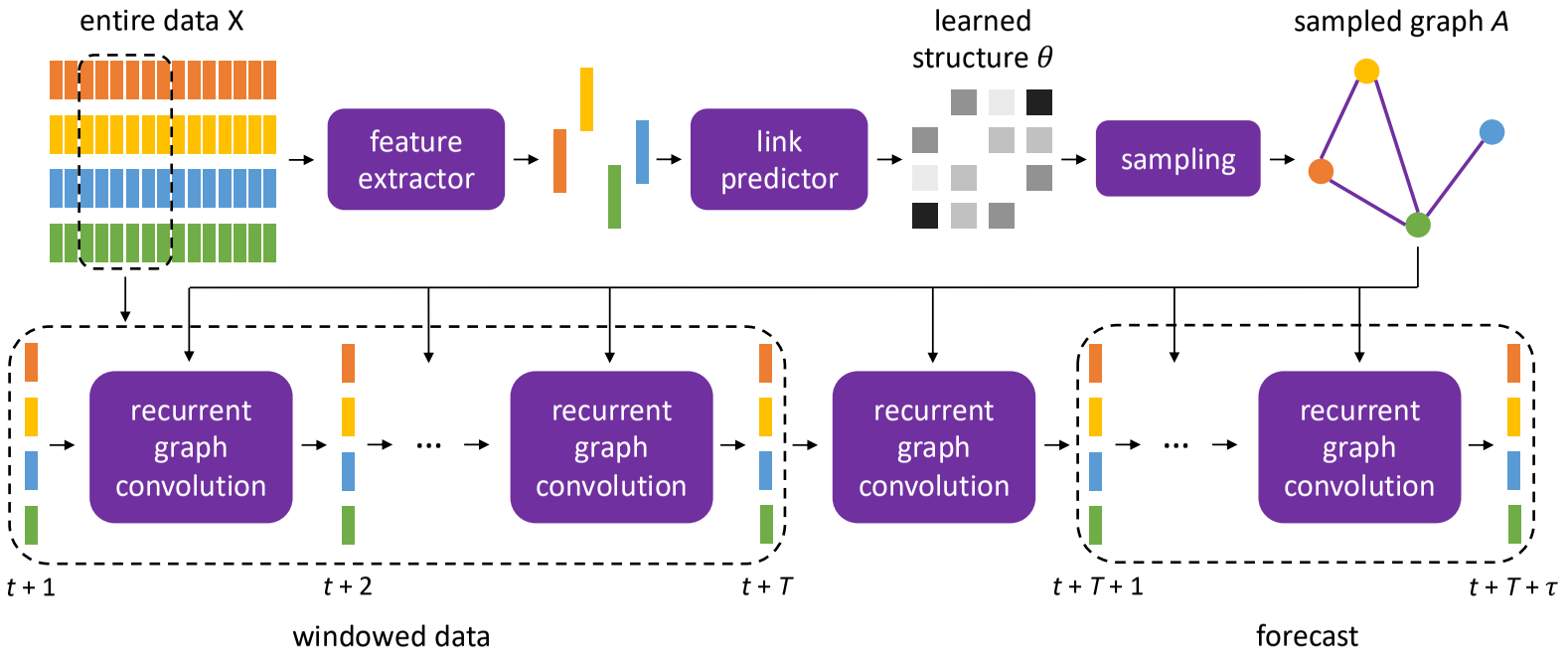


Figure 4.1: GTS architecture.

4.3.2 Graph Neural Network Forecasting

We use a sequence-to-sequence (seq2seq) model [81] to map $X^i_{t+1:t+T}$ to $X^i_{t+T+1:t+T+\tau}$ for each series i. This model is typically a recurrent model, but with a graph structure available among the series, we leverage recurrent graph convolution that handles all series simultaneously, as opposed to the usual recurrent mechanism that treats each series separately.

Specifically, for each time step t', the seq2seq model takes $X_{t'}$ for all series as input and updates the internal hidden state from $H_{t'-1}$ to $H_{t'}$. The encoder part of the seq2seq performs recurrent updates from $t' = t+1$ to $t' = t+T$, producing H_{t+T} as a summary of the input. The decoder part uses H_{t+T} to continue the recurrence and outputs prediction $\widehat{X}_{t'}$ step by step for $t' = t+T+1 : t+T+\tau$.

The recurrence that accepts input and updates hidden states collectively for all series uses a graph convolution to replace the usual multiplication of the input with a weight matrix. Several existing architectures serve this purpose (e.g., GCRN [74], STGCN [96], and T-GCN [101]), but we use the diffusion convolutional GRU defined in DCRNN [51] because it is designed for directed graphs:

$$
\begin{aligned}
R_{t'} &= (W_R \star_A [X_{t'} \parallel H_{t'-1}] + b_R), \\
C_{t'} &= \tanh(W_C \star_A [X_{t'} \parallel (R_{t'} \odot H_{t'-1}] + b_C), \\
U_{t'} &= (W_U \star_A [X_{t'} \parallel H_{t'-1}] + b_U), \\
H_{t'} &= U_{t'} \odot H_{t'-1} + (1 - U_{t'}) \odot C_{t'},
\end{aligned}
$$

where the graph convolution $\star_A$ is defined as

$$W_Q \star_A X = \sum_{k=0}^{K} \left(w_{k,1}^Q (D_O^{-1} A)^k + w_{k,2}^Q (D_I^{-1} A^T)^k \right) X,$$

with D_O and D_I being the out-degree and in-degree matrix. Here, $w_{k,1}^Q$, $w_{k,2}^Q$, b_Q for $Q = R, U, C$ are model parameters and the diffusion degree K is a hyperparemeter.

4.3.3 Training, Optionally with A Priori Knowledge of the Graph

The base training loss (per window) is the mean absolute error between the forecast and the ground truth

$$\ell_{\text{base}}^t(\widehat{X}_{t+T+1:t+T+\tau},\ X_{t+T+1:t+T+\tau}) = \tfrac{1}{\tau} \sum_{t'=t+T+1}^{t+T+\tau} |\widehat{X}_{t'} - X_{t'}|.$$

Additionally, we propose a regularization that improves model accuracy and graph quality, through injecting a priori knowledge of the pairwise interaction into the model. Sometimes an actual graph among the time series is known, such as the case of traffic network mentioned in Section 4.1. Generally, even if an explicit structure is unknown, a neighborhood graph (such as a kNN graph) may still serve as reasonable knowledge. The use of kNN encourages sparsity if k is small, which circumvents the drawback of ℓ_1 constraints that cannot be easily imposed because the graph is not a raw variable to optimize. As such, we use the cross-entropy between θ and the a priori graph A^{a} as the regularization:

$$\ell_{\text{reg}} = \sum_{ij} -A_{ij}^{\text{a}} \log \theta_{ij} - (1 - A_{ij}^{\text{a}}) \log(1 - \theta_{ij}).$$

The overall training loss is then $\sum_t \ell^t_{\text{base}} + \lambda \ell_{\text{reg}}$, with $\lambda > 0$ being the regularization magnitude.

4.3.4 Comparison with NRI

GTS appears similar to NRI [43] on the surface, because both compute a pairwise structure from multiple time series and use the structure to improve forecasting. In these two methods, the architecture to compute the structure, as well as the one to forecast, bare many differences; but these differences are only secondary. The most essential distinction is the number of structures. To avoid confusion, here we say "structure" (θ) rather than "graph" (A) because there are combinatorially many graph samples from the same structure. Our approach produces one single structure given one set of n series. On the contrary, the autoencoder approach adopted by NRI produces different structures given different encoding inputs. Hence, a feasible use of NRI can only occur in the following two manners. (a) A single set of n series is given and training is done on windowed data, where each window will produce a separate structure. (b) Many sets are given and training is done through iterating each set, which corresponds to a separate structure. Both cases are different from our scenario, where a single set of time series is given and a single structure is produced.

4.4 Experiments

In this section, we conduct extensive experiments to show that the proposed method GTS outperforms a comprehensive set of forecasting methods, including one that learns a hidden graph structure (LDS, adapted for time series). We also demon-

strate that GTS is computationally efficient and learns sparse graphs with proper regularization.

4.4.1 Datasets

We experiment with the two benchmark datasets METR-LA and PEMS-BAY from [51].

METR-LA is a traffic dataset collected from loop detectors in the highway of Los Angles, CA [35]. It contains 207 sensors, each of which records four months of data at the frequency of five minutes. A graph of sensors is given; it was constructed by imposing a radial basis function on the pairwise distance of sensors at a certain cutoff. For more information see [51]. We perform no processing and follow the same configuration as in [51] for experimentation.

PEMS-BAY is also a traffic dataset, collected by the California Transportation Agencies Performance Measurement System. It includes 325 sensors in the Bay Area for a period of six months, at the same five-minute frequency. Construction of the graph is the same as that of METR-LA. No processing is performed.

For all datasets, we perform a temporal 70/10/20 split for training, validation, and testing, respectively.

4.4.2 Setup

Baselines. We compare with a number of forecasting methods:

1. Non-deep learning methods: historical average (HA), ARIMA with Kalman filter (ARIMA), vector auto-regression (VAR), and support vector regression (SVR).

The historical average accounts for weekly seasonality and predicts for a day by using the weighted average of the same day in the past few weeks.

2. Deep learning methods that treat each series separately (i.e., no graph): feed-forward neural network (FNN) and LSTM.

3. Graph neural network method applied on the given graph: DCRNN [51].

4. Graph neural network method that simultaneously learns a graph structure: we use LDS [21] to learn the graph, wherein the forecasting model is DCRNN; see Eqn (4.1). We name the method "LDS" for short.

Except LDS, all baselines follow the configurations presented in [51]. For LDS, we follow [21].

Evaluation metrics. All methods are evaluated with three metrics: mean absolute error (MAE), root mean square error (RMSE), and mean absolute percentage error (MAPE).

Hyperparameters. Several hyperparameters are tuned through grid search: initial learning rate {0.1, 0.01, 0.001}, dropout rate {0.1, 0.2, 0.3}, embedding size of LSTM {32, 64, 128, 256}, the k value in kNN {5, 10, 20, 30}, and the weight of regularization {0, 1, 2, 5, 10, 20}. For other hyperparameters, the convolution kernel size in the feature extractor is 10 and the decay ratio of learning rate is 0.1. After tuning, the best initial learning rate for METR-LA and PEMS-BAY is 0.01. The optimizer is Adam.

Because the loss function is an expectation (see (4.1) and (4.2)), the expectation is computed as an average of 10 random samples. Such an averaging is needed only for model evaluation. In training, one random sample suffices because the optimizer

is a stochastic optimizer.

Platform. We implement the models in PyTorch. All experiments are run on one compute node of an IBM Power9 server. The compute node contains 80 CPU cores and four Nvidia V100 GPUs, but because of scheduling limitation we use only one GPU.

4.4.3 Results

Forecasting quality. We first evaluate the performance of GTS through comparing it with all the aforementioned baselines. The tasks are to forecast 15, 30, and 60 minutes.

Table 4.1 summarizes the results for METR-LA. A few observations follow. (1) Deep learning methods generally outperform non-deep learning methods, except historical average that performs on par with deep learning in some metrics. Seasonality is a strong indicator of the repeating traffic patterns and not surprisingly HA performs reasonably well despite simplicity. (2) Among the deep learning methods, graph-based models outperform non-graph models. This result corroborates the premise of this work: graph structure is helpful. (3) Among the graph-based methods, LDS performs slightly better than DCRNN. The difference between these two methods is that the latter employs the given graph, which may or may not imply direct interactions, whereas the former learns a graph in the data-driven manner. Their performances however are quite similar. (4) The most encouraging result is that the proposed method GTS significantly outperforms LDS and hence DCRNN. GTS learns a graph structure through parameterization, rather than treating it a (hyper)parameter as in LDS. This appealing result demonstrates the competitiveness of GTS.

Table 4.1: Forecasting error (METR-LA).

	Metric	HA	ARIMA	VAR	SVR	FNN	LSTM	DCRNN	LDS	GTS
15 min	MAE	4.16	3.99	4.42	3.99	3.99	3.44	2.77	2.75	**2.39**
	RMSE	7.80	8.21	7.89	8.45	7.94	6.30	5.38	5.35	**4.41**
	MAPE	13.0%	9.6%	10.2%	9.3%	9.9%	9.6%	7.3%	7.1%	**6.0%**
30 min	MAE	4.16	5.15	5.41	5.05	4.23	3.77	3.15	3.14	**2.65**
	RMSE	7.80	10.45	9.13	10.87	8.17	7.23	6.45	6.45	**5.06**
	MAPE	13.0%	12.7%	12.7%	12.1%	12.9%	10.9%	8.8%	8.6%	**7.0%**
60 min	MAE	4.16	6.90	6.52	6.72	4.49	4.37	3.60	3.63	**2.99**
	RMSE	7.80	13.23	10.11	13.76	8.69	8.69	7.59	7.67	**5.85**
	MAPE	13.0%	17.4%	15.8%	16.7%	14.0%	13.2%	10.5%	10.34%	**8.3%**

Table 4.2: Forecasting error (PEMS-BAY).

	Metric	HA	ARIMA	VAR	SVR	FNN	LSTM	DCRNN	LDS	GTS
15 min	MAE	2.88	1.62	1.74	1.85	2.20	2.05	1.38	1.33	**1.12**
	RMSE	5.59	3.30	3.16	3.59	4.42	4.19	2.95	2.81	**2.16**
	MAPE	6.8%	3.5%	3.6%	3.8%	5.2%	4.8%	2.9%	2.8%	**2.3%**
30 min	MAE	2.88	2.33	2.32	2.48	2.30	2.20	1.74	1.67	**1.34**
	RMSE	5.59	4.76	4.25	5.18	4.63	4.55	3.97	3.80	**2.74**
	MAPE	6.8%	5.4%	5.0%	5.5%	5.4%	5.2%	3.9%	3.8%	**2.9%**
60 min	MAE	2.88	3.38	2.93	3.28	2.46	2.37	2.07	1.99	**1.58**
	RMSE	5.59	6.50	5.44	7.08	4.98	4.96	4.74	4.59	**3.30**
	MAPE	6.8%	8.3%	6.5%	8.0%	5.9%	5.7%	4.9%	4.8%	**3.6%**

To dive into the behavioral difference of GTS and DCRNN, we plot in Figure 4.2 two forecasting examples. One sees that both methods produce smooth series. In the top example, overall the GTS curve is closer to the moving average of the ground truth than is the DCRNN curve (see e.g., the left part and the U shape). In the bottom example, the GTS curve better captures the sharp dip toward the end of the series. In both examples, there exist several short but deep downward spikes. Such anomalous data are captured by neither methods.

Additionally, the results for PEMS-BAY are summarized in Table 4.2. Observations are similar to those of METR-LA. Our model produces the best prediction in all scenarios and under all metrics.

Computational efficiency. We compare the training costs of the graph-based methods: DCRNN, LDS, and GTS. See Figure 4.3. DCRNN is the most efficient to train, since no graph structure learning is involved. To learn the graph, LDS needs orders of magnitude more time than does DCRNN. Recall that LDS employs a bilevel optimization (4.1), which is computationally highly challenging. In contrast, the proposed method GTS learns the graph structure as a byproduct of the model training (4.2). Its training time is approximately three times of that of DCRNN, a favorable overhead compared with the forbidding cost of LDS.

Learned structures. In the proposed method, the graph structure is an intermediate result of the model. It may or may not appear similar to the native graph (e.g., the traffic network and the power grid). A common pattern is that the most likely graph A from the structure θ is not a planar graph, with much edge crossing. Intuitively, traffic networks and power grids are (nearly) planar graphs because of the spatial configuration of the sensors. Hence, the learned structures are quite different from the native graphs. These structures, however, offer a latent interpretation of

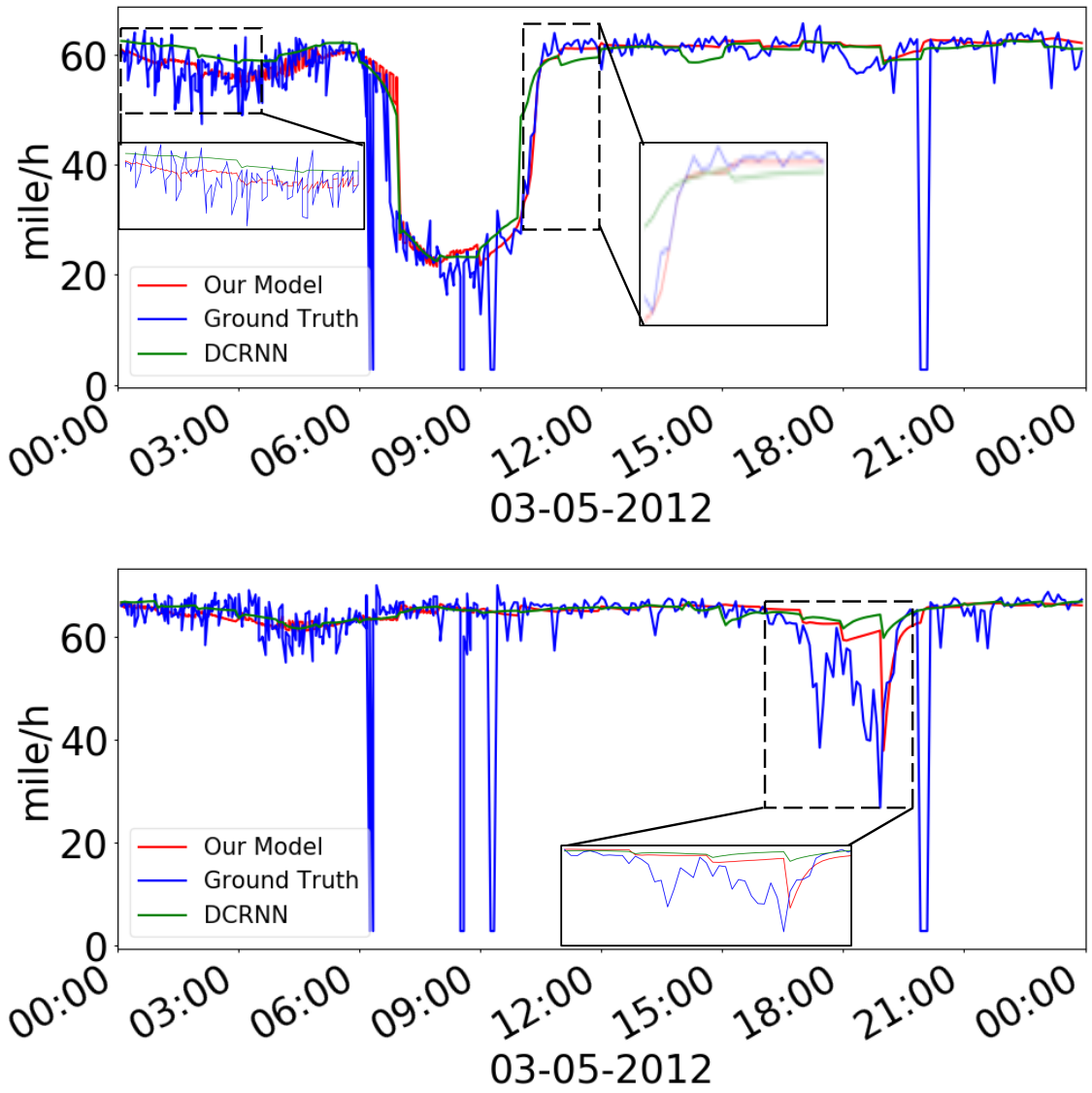


Figure 4.2: One-day forecast (METR-LA).

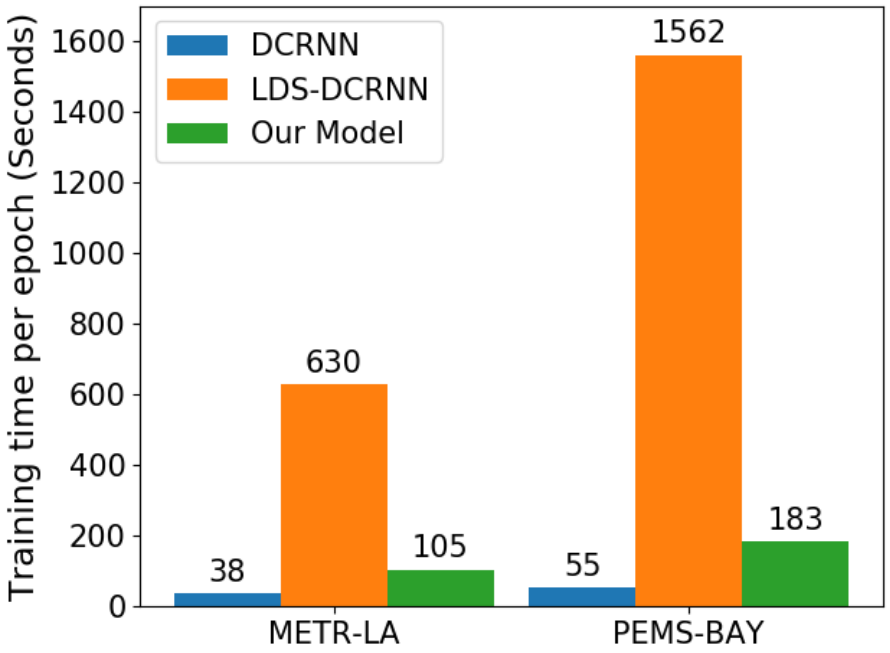
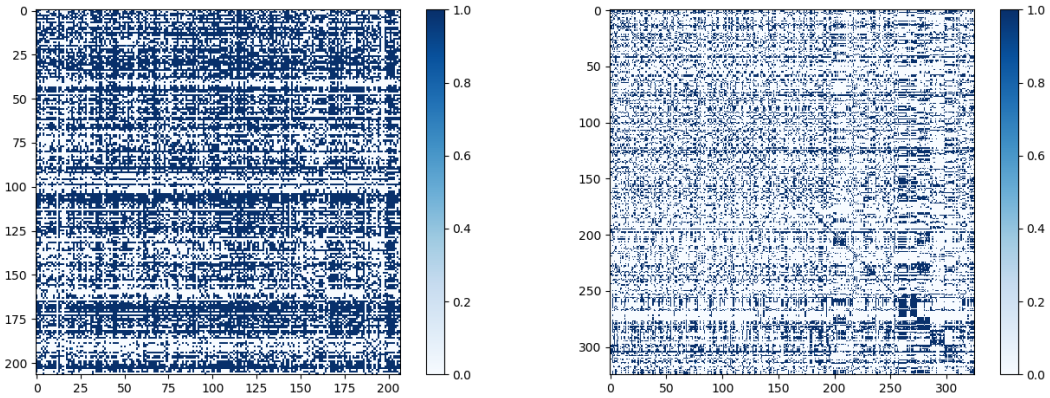


Figure 4.3: Training time per epoch. Not learning a graph (DCRNN) is the fastest to train and learning a graph by using LDS needs orders of magnitude more time. Our model (GTS) learns a graph with a favorable overhead.

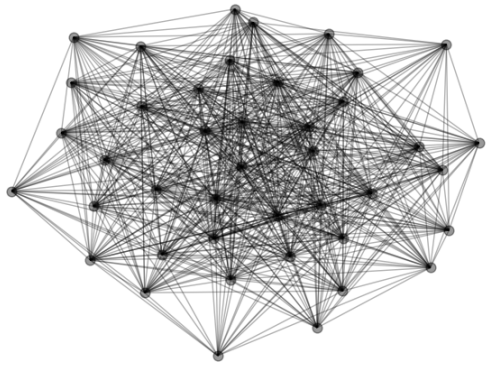
the interaction and when used with graph neural networks, significantly boost their predictive performance.

We plot the learned structure θ for each dataset, corresponding to the 15-minute results in Tables 4.1 and 4.2, in the top row of Figure 4.4. In the two bottom rows we draw the most likely graph A from the structure using a graph layout algorithm that intends to minimize edge crossing. Still, there appears a large number of crossings, indicating that the graphs are far from planar.

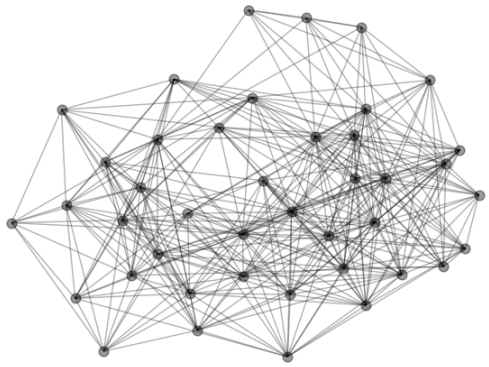


(a) METR-LA

(b) PEMS-BAY



(c) METR-LA (first 42 nodes)



(d) PEMS-BAY (first 42 nodes)

Figure 4.4: Learned structures and graphs.

Chapter 5

Conclusion

Although Graph Convolutional Networks (GCNs) have extended the convolution operation from images to graphs to achieve great performance, there are still many challenges to handle and explore the structure information from a variety of applications. In this dissertation, we propose several structure-aware neural network models to calculate graph convolutions efficiently over the multiple small-scale molecular graphs, a large-scale knowledge graph, and a learned interaction graph. The proposed networks can be trained by an end-to-end training method where a stochastic gradient descent algorithm back-propagates over all network components. The proposed approaches in this dissertation enable new ways to explore the explicit and hidden structure information of many applications.

For the large-scale knowledge graph, we have introduced an end-to-end structure-aware convolutional network (*SACN*). The encoding network is a weighted graph convolutional network, utilizing knowledge graph connectivity structure, node attributes and relation types. *WGCN* with learnable weights has the benefit of collecting adap-

tive amount of information from neighboring graph nodes. In addition, the entity attributes are added as the nodes in the network so that attributes are transformed into knowledge structure information, which is easily integrated into the node embedding. The scoring network of *SACN* is a convolutional neural model, called *Conv-TransE*. It uses a convolutional network to model the relationship as the translation operation and capture the translational characteristic between entities and relations. We also prove that *Conv-TransE* alone has already achieved the state of the art performance. The performance of *SACN* achieves overall about 10% improvement than the state of the art such as *ConvE*. In the future, we would like to incorporate the neighbor selection idea into our training framework, such as, importance pooling in [94] which takes into account the importance of neighbors when aggregating the vector representations of neighbors. We would also like to extend our model to be scalable with larger knowledge graphs encouraged by the results in [94].

For multiple small-scale molecular graphs, a consistent edge-aware multi-view spectral GCN (*EAGCN*) model is proposed. This model builds multiple views connections parameterized by consistent edge attention weights for graph representation and molecular property prediction. Since consistent attention weights from edge mapping dictionaries are shared across all graphs, *EAGCN* model learns the invariant features from graphs and keeps the edge consistency. The proposed spectral convolution operation allowed the more flexible and effective parameterization is proposed. In the experiment, the in-depth analysis of property predictions, molecular graph representations and subtype analysis are provided. In the future, we plan to extend *EAGCN* to multiple types of data in different situations.

For Multivariate time series (MTS) data, most models ignore exploring their interactions between individual time series. Capturing the potential interactions in

time-series models makes them more reliable and reasonable than doing independent models for individual time series. In this dissertation, we explicitly learn the pairwise interactions in the form of a graph and use it to improve forecasting accuracy. We present a time series forecasting model called GTS that learns a graph structure among multiple time series and forecasts them simultaneously with a graph neural network. Hence the proposed GTS model not only discovers the discrete structure but also takes advantage of the generated graph with spatial-temporal graph neural networks. Both the graph generator and the graph neural network are learned end-to-end, maximally exploiting the pairwise interactions among data streams. The graph structure is parameterized by neural networks rather than being treated as a (hyper)parameter, hence significantly reducing the training cost compared with a recently proposed bilevel optimization approach LDS. We conduct comprehensive comparisons with a number of baselines, including non-deep learning methods and deep learning methods (which either ignore the pairwise interaction, use the existing graph, or learn a graph by using LDS), and show that our approach attains the best forecasting quality. We also demonstrate that regularization helps incorporate a apriori knowledge and encourage graph sparsity.

Recently, we attempt to extend our graph learning methods to more domains. A new graph generation task from natural language processing (NLP) caught our attention. Extracting lexico-semantic relations as graph-structured taxonomies, also known as taxonomy construction, has been beneficial in a variety of NLP applications. Recently GNN has shown to be powerful in successfully tackling many tasks. However, there has been no attempt to exploit GNN to create taxonomies. We propose *Graph2Taxo* [75], a GNN-based cross-domain transfer framework for the taxonomy construction task. Our main contribution is to learn the latent features of taxonomy

construction from existing domains to guide the structure learning of an unseen domain. We also propose a novel method of directed acyclic graph (DAG) generation for taxonomy construction. Specifically, our proposed *Graph2Taxo* uses a noisy graph constructed from automatically extracted noisy hyponym-hypernym candidate pairs, and a set of taxonomies for some known domains for training. The learned model is then used to generate taxonomy for a new unknown domain given a set of terms for that domain.

Bibliography

[1] H. Altae-Tran, B. Ramsundar, A. S. Pappu, and V. Pande, "Low data drug discovery with one-shot learning," *ACS central science*, vol. 3, no. 4, pp. 283–293, 2017.

[2] D. Asteriou and S. G. Hall, "ARIMA models and the Box-Jenkins methodology," in *Applied Econometrics*. Palgrave MacMillan, 2011, pp. 265–286.

[3] J. Atwood and D. Towsley, "Diffusion-convolutional neural networks," in *Advances in Neural Information Processing Systems*, 2016, pp. 1993–2001.

[4] S. Auer, C. Bizer, G. Kobilarov, J. Lehmann, R. Cyganiak, and Z. Ives, "Dbpedia: A nucleus for a web of open data," in *The semantic web*. Springer, 2007, pp. 722–735.

[5] L. E. Baum and T. Petrie, "Statistical inference for probabilistic functions of finite state markov chains," *The Annals of Mathematical Statistics*, vol. 37, no. 6, pp. 1554–1563, 1966.

[6] K. Bollacker, C. Evans, P. Paritosh, T. Sturge, and J. Taylor, "Freebase: a collaboratively created graph database for structuring human knowledge," in

Proceedings of the 2008 ACM SIGMOD international conference on Management of data. AcM, 2008, pp. 1247–1250.

[7] A. Bordes, N. Usunier, A. Garcia-Duran, J. Weston, and O. Yakhnenko, "Translating embeddings for modeling multi-relational data," in *Advances in neural information processing systems*, 2013, pp. 2787–2795.

[8] M. M. Bronstein, J. Bruna, Y. LeCun, A. Szlam, and P. Vandergheynst, "Geometric deep learning: going beyond euclidean data," *IEEE Signal Processing Magazine*, vol. 34, no. 4, pp. 18–42, 2017.

[9] J. Bruna and X. Li, "Community detection with graph neural networks," *arXiv preprint arXiv:1705.08415*, 2017.

[10] J. Bruna, W. Zaremba, A. Szlam, and Y. LeCun, "Spectral networks and locally connected networks on graphs," *arXiv preprint arXiv:1312.6203*, 2013.

[11] C. Chieh, "Bond lengths and energies," *University of Waterloo. http://www.science.uwaterloo. ca/˜ cchieh/cact/c120/bondel. html*, 2007.

[12] K. Cho, B. van Merrienboera, C. Gulcehre, D. Bahdanau, F. Bougares, H. Schwenk, and Y. Bengio, "Learning phrase representations using RNN encoder-decoder for statistical machine translation," in *EMNLP*, 2014.

[13] F. R. Chung, *Spectral graph theory*. American Mathematical Soc., 1997, no. 92.

[14] C. W. Coley, R. Barzilay, W. H. Green, T. S. Jaakkola, and K. F. Jensen, "Convolutional embedding of attributed molecular graphs for physical property prediction," *Journal of chemical information and modeling*, vol. 57, no. 8, pp. 1757–1772, 2017.

[15] M. Defferrard, X. Bresson, and P. Vandergheynst, "Convolutional neural networks on graphs with fast localized spectral filtering," in *Advances in Neural Information Processing Systems*, 2016, pp. 3844–3852.

[16] T. Dettmers, P. Minervini, P. Stenetorp, and S. Riedel, "Convolutional 2d knowledge graph embeddings," *arXiv preprint arXiv:1707.01476*, 2017.

[17] J. Devlin, M.-W. Chang, K. Lee, and K. Toutanova, "BERT: Pre-training of deep bidirectional transformers for language understanding," in *NAACL-HLT*, 2019.

[18] D. K. Duvenaud, D. Maclaurin, J. Iparraguirre, R. Bombarell, T. Hirzel, A. Aspuru-Guzik, and R. P. Adams, "Convolutional networks on graphs for learning molecular fingerprints," in *Advances in neural information processing systems*, 2015, pp. 2224–2232.

[19] A. Fout, J. Byrd, B. Shariat, and A. Ben-Hur, "Protein interface prediction using graph convolutional networks," in *Advances in Neural Information Processing Systems*, 2017, pp. 6533–6542.

[20] L. Franceschi, M. Donini, P. Frasconi, and M. Pontil, "Forward and reverse gradient-based hyperparameter optimization," in *ICML*, 2017.

[21] L. Franceschi, M. Niepert, M. Pontil, and X. He, "Learning discrete structures for graph neural networks," in *ICML*, 2019.

[22] S. Fu, W. Liu, Y. Zhou, and L. Nie, "Hplapgcn: Hypergraph p-laplacian graph convolutional networks," *Neurocomputing*, vol. 362, pp. 166–174, 2019.

[23] J. Gilmer, S. S. Schoenholz, P. F. Riley, O. Vinyals, and G. E. Dahl, "Neural message passing for quantum chemistry," *arXiv preprint arXiv:1704.01212*, 2017.

[24] R. C. Glen, A. Bender, C. H. Arnby, L. Carlsson, S. Boyer, and J. Smith, "Circular fingerprints: flexible molecular descriptors with applications from physical chemistry to adme," *IDrugs*, vol. 9, no. 3, p. 199, 2006.

[25] R. Gómez-Bombarelli, J. N. Wei, D. Duvenaud, J. M. Hernández-Lobato, B. Sánchez-Lengeling, D. Sheberla, J. Aguilera-Iparraguirre, T. D. Hirzel, R. P. Adams, and A. Aspuru-Guzik, "Automatic chemical design using a data-driven continuous representation of molecules," *ACS central science*, vol. 4, no. 2, pp. 268–276, 2018.

[26] P. Goyal and E. Ferrara, "Graph embedding techniques, applications, and performance: A survey," *Knowledge-Based Systems*, vol. 151, pp. 78–94, 2018.

[27] C. W. J. Granger, "Investigating causal relations by econometric models and cross-spectral methods," *Econometrica*, vol. 37, no. 3, pp. 424–438, 1969.

[28] A. Grover and J. Leskovec, "node2vec: Scalable feature learning for networks," in *Proceedings of the 22nd ACM SIGKDD international conference on Knowledge discovery and data mining*. ACM, 2016, pp. 855–864.

[29] J. D. Hamilton, *Time Series Analysis*. Princeton University Press, 1994.

[30] W. Hamilton, Z. Ying, and J. Leskovec, "Inductive representation learning on large graphs," in *Advances in Neural Information Processing Systems*, 2017, pp. 1025–1035.

[31] W. L. Hamilton, R. Ying, and J. Leskovec, "Representation learning on graphs: Methods and applications," *arXiv preprint arXiv:1709.05584*, 2017.

[32] M. Henaff, J. Bruna, and Y. LeCun, "Deep convolutional networks on graph-structured data," *arXiv preprint arXiv:1506.05163*, 2015.

[33] S. Hochreiter and J. Schmidhuber, "Long short-term memory," *Neural Computation*, vol. 9, no. 8, pp. 1735–1780, 1997.

[34] M. L. Huggins, "Bond energies and polarities1," *Journal of the American Chemical Society*, vol. 75, no. 17, pp. 4123–4126, 1953.

[35] H. V. Jagadish, J. Gehrke, A. Labrinidis, Y. Papakonstantinou, J. M. Patel, R. Ramakrishnan, and C. Shahabi, "Big data and its technical challenges," *Commun. ACM*, vol. 57, no. 7, pp. 86–94, 2014.

[36] E. Jang, S. Gu, and B. Poole, "Categorical reparameterization with Gumbel-Softmax," in *ICLR*, 2017.

[37] G. Ji, S. He, L. Xu, K. Liu, and J. Zhao, "Knowledge graph embedding via dynamic mapping matrix," in *Proceedings of the 53rd Annual Meeting of the Association for Computational Linguistics and the 7th International Joint Conference on Natural Language Processing (Volume 1: Long Papers)*, vol. 1, 2015, pp. 687–696.

[38] S. Kearnes, K. McCloskey, M. Berndl, V. Pande, and P. Riley, "Molecular graph convolutions: moving beyond fingerprints," *Journal of computer-aided molecular design*, vol. 30, no. 8, pp. 595–608, 2016.

[39] N. Khan, U. Chaudhuri, B. Banerjee, and S. Chaudhuri, "Graph convolutional network for multi-label vhr remote sensing scene recognition," *Neurocomputing*, vol. 357, pp. 36–46, 2019.

[40] R. Khasanova and P. Frossard, "Graph-based isometry invariant representation learning," *arXiv preprint arXiv:1703.00356*, 2017.

[41] D. P. Kingma and J. Ba, "Adam: A method for stochastic optimization," *arXiv preprint arXiv:1412.6980*, 2014.

[42] T. N. Kipf and M. Welling, "Variational graph auto-encoders," in *Advances in neural information processing systems,Bayesian Deep Learning Workshop*, 2016.

[43] T. Kipf, E. Fetaya, K.-C. Wang, M. Welling, and R. Zemel, "Neural relational inference for interacting systems," in *ICML*, 2018.

[44] T. N. Kipf and M. Welling, "Semi-supervised classification with graph convolutional networks," *arXiv preprint arXiv:1609.02907*, 2016.

[45] S. Lachapelle, P. Brouillard, T. Deleu, and S. Lacoste-Julien, "Gradient-based neural DAG learning," in *ICLR*, 2020.

[46] Y. LeCun, Y. Bengio, and G. Hinton, "Deep learning," *nature*, vol. 521, no. 7553, p. 436, 2015.

[47] Y. LeCun, L. Bottou, Y. Bengio, and P. Haffner, "Gradient-based learning applied to document recognition," *Proceedings of the IEEE*, vol. 86, no. 11, pp. 2278–2324, 1998.

[48] J. B. Lee, R. A. Rossi, S. Kim, N. K. Ahmed, and E. Koh, "Attention models in graphs: A survey," *arXiv preprint arXiv:1807.07984*, 2018.

[49] R. Levie, F. Monti, X. Bresson, and M. M. Bronstein, "Cayleynets: Graph convolutional neural networks with complex rational spectral filters," *arXiv preprint arXiv:1705.07664*, 2017.

[50] R. Li, S. Wang, F. Zhu, and J. Huang, "Adaptive graph convolutional neural networks," *arXiv preprint arXiv:1801.03226*, 2018.

[51] Y. Li, R. Yu, C. Shahabi, and Y. Liu, "Diffusion convolutional recurrent neural network: Data-driven traffic forecasting," in *ICLR*, 2018.

[52] D. R. Lide, *CRC handbook of chemistry and physics: a ready-reference book of chemical and physical data*. CRC press, 1995.

[53] Y. Lin, Z. Liu, and M. Sun, "Knowledge representation learning with entities, attributes and relations," *ethnicity*, vol. 1, pp. 41–52, 2016.

[54] Y. Lin, Z. Liu, M. Sun, Y. Liu, and X. Zhu, "Learning entity and relation embeddings for knowledge graph completion." in *AAAI*, vol. 15, 2015, pp. 2181–2187.

[55] A. Lusci, G. Pollastri, and P. Baldi, "Deep architectures and deep learning in chemoinformatics: the prediction of aqueous solubility for drug-like molecules," *Journal of chemical information and modeling*, vol. 53, no. 7, pp. 1563–1575, 2013.

[56] L. v. d. Maaten and G. Hinton, "Visualizing data using t-sne," *Journal of machine learning research*, vol. 9, no. Nov, pp. 2579–2605, 2008.

[57] C. J. Maddison, A. Mnih, and Y. W. Teh, "The concrete distribution: A continuous relaxation of discrete random variables," in *ICLR*, 2017.

[58] F. Mahdisoltani, J. Biega, and F. M. Suchanek, "Yago3: A knowledge base from multilingual wikipedias," in *CIDR*, 2013.

[59] S. G. Makridakis, S. C. Wheelwright, and R. J. Hyndman, *Forecasting: Methods and Applications*, 3rd ed. Wiley, 1997.

[60] R. S. Michalski, J. G. Carbonell, and T. M. Mitchell, *Machine learning: An artificial intelligence approach*. Springer Science & Business Media, 2013.

[61] T. Mikolov, K. Chen, G. Corrado, and J. Dean, "Efficient estimation of word representations in vector space," *arXiv preprint arXiv:1301.3781*, 2013.

[62] D. L. Mobley and J. P. Guthrie, "Freesolv: a database of experimental and calculated hydration free energies, with input files," *Journal of computer-aided molecular design*, vol. 28, no. 7, pp. 711–720, 2014.

[63] F. Monti, D. Boscaini, J. Masci, E. Rodola, J. Svoboda, and M. M. Bronstein, "Geometric deep learning on graphs and manifolds using mixture model cnns," in *Proc. CVPR*, vol. 1, no. 2, 2017, p. 3.

[64] D. Q. Nguyen, T. D. Nguyen, D. Q. Nguyen, and D. Phung, "A novel embedding model for knowledge base completion based on convolutional neural network," *arXiv preprint arXiv:1712.02121*, 2017.

[65] D. Q. Nguyen, "An overview of embedding models of entities and relationships for knowledge base completion," *arXiv preprint arXiv:1703.08098*, 2017.

[66] D. Q. Nguyen, K. Sirts, L. Qu, and M. Johnson, "Stranse: a novel embedding model of entities and relationships in knowledge bases," *arXiv preprint arXiv:1606.08140*, 2016.

[67] M. Niepert, M. Ahmed, and K. Kutzkov, "Learning convolutional neural networks for graphs," in *International conference on machine learning*, 2016, pp. 2014–2023.

[68] T. Niwa, B.-W. Ying, K. Saito, W. Jin, S. Takada, T. Ueda, and H. Taguchi, "Bimodal protein solubility distribution revealed by an aggregation analysis of the entire ensemble of escherichia coli proteins," *Proceedings of the National Academy of Sciences*, vol. 106, no. 11, pp. 4201–4206, 2009.

[69] A. G. Orpen, L. Brammer, F. H. Allen, O. Kennard, D. G. Watson, and R. Taylor, "Supplement. tables of bond lengths determined by x-ray and neutron diffraction. part 2. organometallic compounds and co-ordination complexes of the d-and f-block metals," *Journal of the Chemical Society, Dalton Transactions*, no. 12, pp. S1–S83, 1989.

[70] T. Pham, T. Tran, D. Phung, and S. Venkatesh, "Column networks for collective classification," 2017.

[71] D. Rogers and M. Hahn, "Extended-connectivity fingerprints," *Journal of chemical information and modeling*, vol. 50, no. 5, pp. 742–754, 2010.

[72] M. Schlichtkrull, T. N. Kipf, P. Bloem, R. van den Berg, I. Titov, and M. Welling, "Modeling relational data with graph convolutional networks," in *European Semantic Web Conference*. Springer, 2018, pp. 593–607.

[73] M. Schuster and K. K. Paliwal, "Bidirectional recurrent neural networks," *IEEE Transactions on Signal Processing*, vol. 45, no. 11, pp. 2673–2681, 1997.

[74] Y. Seo, M. Defferrard, P. Vandergheynst, and X. Bresson, "Structured sequence modeling with graph convolutional recurrent networks," arXiv:1612.07659, 2016.

[75] C. Shang, S. Dash, M. F. M. Chowdhury, N. Mihindukulasooriya, and A. Gliozzo, "Taxonomy construction of unseen domains via graph-based cross-domain knowledge transfer," in *Proceedings of the 58th Annual Meeting of the Association for Computational Linguistics*, 2020, pp. 2198–2208.

[76] C. Shang, Q. Liu, K.-S. Chen, J. Sun, J. Lu, J. Yi, and J. Bi, "Edge attention-based multi-relational graph convolutional networks," *arXiv*, pp. arXiv–1802, 2018.

[77] C. Shang, Y. Tang, J. Huang, J. Bi, X. He, and B. Zhou, "End-to-end structure-aware convolutional networks for knowledge base completion," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 33, 2019, pp. 3060–3067.

[78] A. Sharma, P. Talukdar *et al.*, "Towards understanding the geometry of knowledge graph embeddings," in *Proceedings of the 56th Annual Meeting of the Association for Computational Linguistics (Volume 1: Long Papers)*, vol. 1, 2018, pp. 122–131.

[79] M. Simonovsky and N. Komodakis, "Dynamic edge-conditioned filters in convolutional neural networks on graphs."

[80] A. Susnjara, N. Perraudin, D. Kressner, and P. Vandergheynst, "Accelerated filtering on graphs using lanczos method," *arXiv preprint arXiv:1509.04537*, 2015.

[81] I. Sutskever, O. Vinyals, and Q. V. Le, "Sequence to sequence learning with neural networks," in *NIPS*, 2014.

[82] K. Toutanova and D. Chen, "Observed versus latent features for knowledge base and text inference," in *Proceedings of the 3rd Workshop on Continuous Vector Space Models and their Compositionality*, 2015, pp. 57–66.

[83] T. Trouillon, J. Welbl, S. Riedel, É. Gaussier, and G. Bouchard, "Complex embeddings for simple link prediction," in *International Conference on Machine Learning*, 2016, pp. 2071–2080.

[84] A. Vaswani, N. Shazeer, N. Parmar, J. Uszkoreit, L. Jones, A. N. Gomez, L. Kaiser, and I. Polosukhin, "Attention is all you need," in *NIPS*, 2017.

[85] P. Veličković, G. Cucurull, A. Casanova, A. Romero, P. Liò, and Y. Bengio, "Graph attention networks," *arXiv preprint arXiv:1710.10903*, 2017.

[86] Q. Wang, Z. Mao, B. Wang, and L. Guo, "Knowledge graph embedding: A survey of approaches and applications," *IEEE Transactions on Knowledge and Data Engineering*, vol. 29, no. 12, pp. 2724–2743, 2017.

[87] Z. Wang, J. Zhang, J. Feng, and Z. Chen, "Knowledge graph embedding by translating on hyperplanes." in *AAAI*, vol. 14, 2014, pp. 1112–1119.

[88] R. J. Williams, G. E. Hinton, and D. E. Rumelhart, "Learning representations by back-propagating errors," *Nature*, vol. 323, no. 6088, pp. 533–536, 1986.

[89] Z. Wu, B. Ramsundar, E. N. Feinberg, J. Gomes, C. Geniesse, A. S. Pappu, K. Leswing, and V. Pande, "Moleculenet: a benchmark for molecular machine learning," *Chemical Science*, vol. 9, no. 2, pp. 513–530, 2018.

[90] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and S. Y. Philip, "A comprehensive survey on graph neural networks," *IEEE Transactions on Neural Networks and Learning Systems*, 2020.

[91] Z. Wu, S. Pan, F. Chen, G. Long, C. Zhang, and P. S. Yu, "A comprehensive survey on graph neural networks," Preprint arXiv:1901.00596, 2019.

[92] K. Xu, W. Hu, J. Leskovec, and S. Jegelka, "How powerful are graph neural networks?" in *International Conference on Learning Representations*, 2018.

[93] B. Yang, W.-t. Yih, X. He, J. Gao, and L. Deng, "Embedding entities and relations for learning and inference in knowledge bases," *arXiv preprint arXiv:1412.6575*, 2014.

[94] R. Ying, R. He, K. Chen, P. Eksombatchai, W. L. Hamilton, and J. Leskovec, "Graph convolutional neural networks for web-scale recommender systems," in *Proceedings of the 24th ACM SIGKDD International Conference on Knowledge Discovery Data Mining*, 2018, pp. 974–983.

[95] Z. Ying, J. You, C. Morris, X. Ren, W. Hamilton, and J. Leskovec, "Hierarchical graph representation learning with differentiable pooling," in *Advances in Neural Information Processing Systems*, 2018.

[96] B. Yu, H. Yin, and Z. Zhu, "Spatio-temporal graph convolutional networks: A deep learning framework for traffic forecasting," in *IJCAI*, 2018.

[97] Y. Yu, J. Chen, T. Gao, and M. Yu, "DAG-GNN: DAG structure learning with graph neural networks," in *ICML*, 2019.

[98] P. Zarchan and H. Musoff, *Fundamentals of Kalman Filtering*. American Institute of Aeronautics and Astronautics, Incorporated, 2000.

[99] Z. Zhang, P. Cui, and W. Zhu, "Deep learning on graphs: A survey," Preprint arXiv:1812.04202, 2018.

[100] ——, "Deep learning on graphs: A survey," *IEEE Transactions on Knowledge and Data Engineering*, 2020.

[101] L. Zhao, Y. Song, C. Zhang, Y. Liu, P. Wang, T. Lin, M. Deng, and H. Li, "T-GCN: A temporal graph convolutional network for traffic prediction," *IEEE Transactions on Intelligent Transportation Systems*, 2019.

[102] X. Zheng, B. Aragam, P. Ravikumar, and E. P. Xing, "DAGs with NO TEARS: Continuous optimization for structure learning," in *NeurIPS*, 2018.

[103] J. Zhou, G. Cui, Z. Zhang, C. Yang, Z. Liu, L. Wang, C. Li, and M. Sun, "Graph neural networks: A review of methods and applications," Preprint arXiv:1812.08434, 2018.